PCT/GB 98/03632

09/555465

GB98/03632

The Patent Office

The Patent Office
Concept House
Cardiff Road
Newport
South Wales
NP9 1RH

22 DEC 1998
WIPO PCT

E.A.J.U.

PRIORITY DOCUMENT

SUBMITTED OR TRANSMITTED IN COMPLIANCE WITH RULE 17.1(a) OR (b)

I, the undersigned, being an officer duly authorised in accordance with Section 74(1) and (4) of the Deregulation & Contracting Out Act 1994, to sign and issue certificates on behalf of the Comptroller-General, hereby certify that annexed hereto is a true copy of the documents as originally filed in connection with the patent application identified therein.

In accordance with the Patents (Companies Re-registration) Rules 1982, if a company named in this certificate and any accompanying documents has re-registered under the Companies Act 1980 with the same name as that with which it was registered immediately before re-registration save for the substitution as, or inclusion as, the last part of the name of the words "public limited company" or their equivalents in Welsh, references to the name of the company in this certificate and any accompanying documents shall be treated as references to the name with which it is so re-registered.

In accordance with the rules, the words "public limited company" may be replaced by p.l.c., plc, P.L.C. or PLC.

Re-registration under the Companies Act does not constitute a new legal entity but merely subjects the company to certain additional company law rules.

Signed

Dated 12/8/98

Patents Form 1/77

Pat[redacted] Act 1977
(Rule 16)

The Patent Office

THE PATENT OFFICE
K
-4 DEC 1997
RECEIVED BY HAND

1/77

05DEC97 E322464-1 D01463
P01/7700 25.00 - 9725742.2

Request for grant of a patent

(See the notes on the back of this form. You can also get an explanatory leaflet from the Patent Office to help you fill in this form)

The Patent Office

Cardiff Road
Newport
Gwent NP9 1RH

1. Your reference

30970139 UK

2. Patent application nu[redacted]
(The Patent Office will fill

9725742.2

04 DEC 1997

3. Full name, address and postcode of the or of each applicant *(underline all surnames)*

Hewlett-Packard Company
3000 Hanover Street
Palo Alto
California 94304
United States of America

Patents ADP number *(if you know it)*

If the applicant is a corporate body, give the country/state of its incorporation

California USA

4965880001

4. Title of the invention

OBJECT GATEWAY

5. Name of your agent *(if you have one)*

LAWMAN, MATTHEW JOHN

"Address for service" in the United Kingdom to which all correspondence should be sent *(including the postcode)*

Hewlett-Packard Limited
IP Section, Building 2
Filton Road
Stoke Gifford
Bristol BS12 6QZ

Patents ADP number *(if you know it)*

7337009001

6. If you are declaring priority from one or more earlier patent applications, give the country and the date of filing of the or of each of these earlier applications and *(if you know it)* the or each application number

Country	Priority application number *(if you know it)*	Date of filing *(day / month / year)*

7. If this application is divided or otherwise derived from an earlier UK application, give the number and the filing date of the earlier application

Number of earlier application	Date of filing *(day / month / year)*

8. Is a statement of inventorship and of right to grant of a patent required in support of this request? *(Answer 'Yes' if:*

a) any applicant named in part 3 is not an inventor, or YES
b) there is an inventor who is not named as an applicant, or
c) any named applicant is a corporate body.
See note (d))

9. Enter the number of sheets for any of the following items you are filing with this form. Do not count copies of the same document

Continuation sheets of this form	
Description	62 ✓
Claim(s)	8 ✓
Abstract	a
Drawing(s)	6+6 ✓

10. If you are also filing any of the following, state how many against each item.

Priority documents	
Translations of priority documents	
Statement of inventorship and right to grant of a patent *(Patents Form 7/77)*	
Request for preliminary examination and search *(Patents Form 9/77)*	
Request for substantive examination *(Patents Form 10/77)*	
Any other documents *(please specify)*	

11. I/We request the grant of a patent on the basis of this application.

Signature M. J. Lawman Date 4/12/97

Matthew John Lawman

12. Name and daytime telephone number of person to contact in the United Kingdom: Janet Smith 0117-922-8026

Warning

After an application for a patent has been filed, the Comptroller of the Patent Office will consider whether publication or communication of the invention should be prohibited or restricted under Section 22 of the Patents Act 1977. You will be informed if it is necessary to prohibit or restrict your invention in this way. Furthermore, if you live in the United Kingdom, Section 23 of the Patents Act 1977 stops you from applying for a patent abroad without first getting written permission from the Patent Office unless an application has been filed at least 6 weeks beforehand in the United Kingdom for a patent for the same invention and either no direction prohibiting publication or communication has been given, or any such direction has been revoked.

Notes

a) If you need help to fill in this form or you have any questions, please contact the Patent Office on 0645 500505.

b) Write your answers in capital letters using black ink or you may type them.

c) If there is not enough space for all the relevant details on any part of this form, please continue on a separate sheet of paper and write "see continuation sheet" in the relevant part(s). Any continuation sheet should be attached to this form.

d) If you have answered 'Yes' Patents Form 7/77 will need to be filed.

e) Once you have filled in the form you must remember to sign and date it.

f) For details of the fee and ways to pay please contact the Patent Office.

OBJECT GATEWAY

(30970139)

Technical Field

The present invention relates to distributed computing system and particularly, but not exclusively, to object based, distributed computing systems.

Background Art

Object based systems are generally known. Presently, the best known model for an object-based system, or architecture, is the Common Object Request Broker Architecture (CORBA), which is supported by a consortium called the Object Management Group (OMG) comprising over 700 companies. Another well known model is Microsoft's Distributed Component Object Model (DCOM), which shares many principles with CORBA. While many of the principles discussed below relate to object-based systems in general, for ease of description only, the description will be directed to the specific case of CORBA. The skilled person will appreciate, however, that the principles are more widely applicable, for example to DCOM.

CORBA can be considered as a model for middleware applications, which defines how blocks of intelligence, known as objects, interact across a distributed computing and communications environment. CORBA is well documented and will not be considered in any depth in this description. The book "Instant CORBA" by Robert Orfali et al (published by Wiley Computer Publishing, John Wiley & Sons Inc., Copyright 1997, ISBN 0-471-18333-4), however, provides an in depth introduction into CORBA and, as such, provides background to the principles described herein.

It is usual for companies to protect their internal network, or intranet, from attacks from the public Internet by using gateways or firewalls which restrict and monitor the flow of information between the Internet and intranet. Some companies also have a need to deploy firewalls internally to restrict and monitor the flow of information between different parts of the enterprise.

CORBA is receiving increasing attention as more and more companies adopt it as the model for the backbone for their enterprise information architectures. One of the main barriers to CORBA deployment, however, is the lack of a defined mechanism for implementing gateway or firewall technology.

Hereafter, and purely for the sake of convenience, firewalls, gateways, packet filters and other forms of barrier that can be put in place between the Internet and intranets will all be termed "gateways".

It is an object of the present invention to provide a gateway which can support CORBA and the like.

Disclosure of the Invention

In accordance with a first aspect, the present invention provides a gateway for being situated between a first network and a second network, comprising:

first interface means for receiving from the first network a message (m) intended for an object in the second network;

interceptor means for detecting any object reference(s) in the message (m) and mapping the intercepted object reference(s) to a modified form; and

second interface means for forwarding to the second network a message (m') including any modified object reference(s).

This aspect of the invention addresses a problem that object invocations can include object references to other objects. An object invoked from across the gateway can receive such object references and, subsequently, use them to invoke the respective referenced objects, or pass the references somewhere else. Typically, gateways require a specific proxy to be in place to manage interaction with a particular object across the gateway. Therefore, the gateway needs to include a mechanism to intercept object references in invocations, or in other messages, if communications across the gateway with the object referenced are to be supported at all. Without such a mechanism, communications across the gateway with referenced objects would require that the gateway is pre-prepared with proxies for all possible object invocations.

In a preferred embodiment, the interceptor means is configured for mapping an object reference to a modified object reference which includes both the intercepted object reference and an object reference to respective interface means associated with the gateway.

Thus, the invoked object receives a modified object reference rather than the original object reference. If the invoked object, acting now as an invoking object, uses the modified object reference to make a further object invocation, the invoking object knows to send the invocation to the specified interface means on the gateway. Otherwise, the invoking object would attempt to invoke the referenced object directly, and fail due to the lack of a defined interface means to process the invocation. Further, since the modified object reference also

includes the original object reference, and this information is passed in the invocation, the gateway knows to where the invocation should go.

Preferably, in the case of CORBA, object references are in the form of Interoperable Object References (IORs) including host, port and key parameters. The host parameter defines to which host an invocation of the object should be directed, the port parameter defines to which port on the host the invocation should be directed and the key contains data to be passed to the object when it is invoked.

A modified object reference preferably is in the form of an IOR and has host and port parameters which define the gateway interface means as the host to which the invocation should be directed. The key parameter of the modified object reference includes the original object reference, in the form of an IOR and the gateway, on receipt of an invocation that includes the key, interprets the key and directs the invocation to the referenced object.

In a preferred embodiment, the interceptor means is configured for incorporating into a modified object reference an identifier to indicate from which network the object reference originated. Such an identifier can be used by a gateway receiving an object reference, which is itself a modified object reference, to determine whether the gateway was the one responsible for forming the modified object reference in the first place. If it was, then, rather than modifying the object reference again, the gateway can simply 'unmodify' the reference. This is achieved by forwarding only the original object reference, which is for example extracted from the object key of the modified object reference, to the desired destination.

In a preferred embodiment, the interceptor means is configured for incorporating into a modified object reference a name tag associated with the identity of the interceptor means. The name tag can be used by the gateway to determine whether the object to be invoked is available from the gateway. In a CORBA system, determination of the availability of the object can be made using a Naming Service call, as will be described below.

In a preferred embodiment, the interceptor means is configured for incorporating into a modified object reference check data which is representative of at least a part of the modified object reference.

The check data preferably comprises the result of a hash operation on at least the object reference, host and port, and a secret. The check data is used by the gateway to verify that an invocation has arisen from a valid modified object reference previously generated by the gateway, rather than from an untrusted source. Thus, the check data adds a level of security to the gateway. In this way, even if an untrusted party gains knowledge of the location of a valid object on the other side of the gateway, the party will not be able to invoke

the object without knowing the hash operation and, more particularly, the secret, which is stored securely by the gateway.

In accordance with a second aspect, the present invention provides a gateway for being situated between a first network and a second network, comprising:

first interface means for receiving from the second network a message (n);

interceptor means for detecting any object reference(s) in the message (n); and

second interface means for forwarding to referenced object(s) in the first network data extracted from the respective object reference(s) and intended for the referenced object(s).

This aspect of the invention relates in general to an invocation operation, across the gateway, made on the basis of a previously modified object reference.

This aspect of the invention preferably further includes some or all of features corresponding to those described in relation to the first aspect of the invention, and such further features are claimed in the claims at the end of the present description.

In a preferred embodiment, the gateway further comprising interceptor activating means which is configured for determining, on the basis of an object reference, whether there exists a respective interceptor means for a received object reference.

Thus, an interceptor means to process a particular object reference need not be permanently running in the gateway. In other words, the interceptor means need only be invoked when a respective object reference requiring the interceptor means is received by the gateway. Then, the gateway can initiate the dynamic generation of the interceptor means. In this way, the gateway can minimise state build-up, where sate build-up has some disadvantages, which are described in more detail below. Details of how to generate interceptor means dynamically are also provided below.

In some embodiments, the delay caused in generating dynamic interceptor means can be mitigated by enacting the interceptor means generation before an actual invocation is received. In effect, the generation could occur at any time from the receipt and processing of the object reference (which is used in the eventual invocation) up to the time when the invocation is received. A particularly advantageous time for generation would be as soon as, or just after, the object reference is received. Of course, if this practice were left unmanaged, interceptor means would be generated for every object reference received, and state-build would occur.

A very convenient way to control the state build up would, for example, be to allocate the interceptor means with some form of time-out, for example an hour, 24 hours, or one week, after which the interceptor means would shut down. Before shutting down, the

interceptor means would be instantly available to respective object invocations, thereby speeding up the invocation process across the gateway. After shutting down, the interceptor means would not be instantly available, but would still be available, with a delay, using dynamic interceptor means generation.

It is emphasised that dynamic interceptor, or proxy, generation is applicable to other object-based models, and, in particular, to DCOM.

In a preferred embodiment of the first or the second aspect of the present invention, the gateway is configured for operation within a trusted operating system environment, which supports mandatory access control (MAC). Using MAC can provide greater security for the gateway.

A particularly suitable trusted operating system is the HP-UX 10.09.01 Compartmented Mode Workstation (CMW) sold by Hewlett-Packard Company, which provides a MAC policy governing the way data may be accessed on a trusted system.

The MAC policy is a computerised version of the US Department of Defence's long-standing multilevel security policy for handling classified information. The MAC policy uses labels that reflect information sensitivity, and maintains those labels for every process and file system object to prevent users not cleared for certain levels of classified information from accessing it. Under MAC, users and processes are also assigned clearances. A clearance defines the maximum sensitivity label the user or process can access, which is necessary since some users and processes have privileges that allow them to switch between sensitivity labels. Using the MAC policy, the operating system controls access based on the relative sensitivity of the applications running and the files they access. The HP-UX CMW operating system rates as a B1 grade secure operating system, according to the Orange Book [NCSC] criteria. In general B1 and higher-grade operating systems apply some form of MAC.

The HP-UX 10.09.01 CMW [DIA 91], is described in detail in the documents referenced at the end of this description, which are available from Hewlett-Packard Company. At the time of writing this description, HP-UX 10.09.01 CMW is the current version of the operating system. Future versions of the operating system, and the respective documentation, will, however, remain relevant to the present description and embodiment.

Further details of how to implement the gateway in a trusted operating system such as HP's CMW are provided below.

Preferably the gateway includes a trusted relay process which has the privileges necessary to pass messages between an inside compartment and an outside compartment of the trusted operating system, and wherein messages associated with a network inside of the

gateway and the interceptor means are associated with the inside compartment, and messages associated with a network outside of the gateway are associated with the outside compartment.

The trusted relay process preferably comprises the first interface means or the second interface means depending on which of the first or second networks is the inside network.

It will be appreciated that the interceptor means can receive messages including object references from either the inside network or the outside network. In either case, the interceptor means can map an object reference to a modified object reference, and forward it on.

Preferably, the interceptor means applies a different mapping depending on which direction the object reference is travelling. In particular, each mapping should reflect the specific identifier which indicates from which network the object reference originated.

In accordance with further aspects, the present invention provides methods of passing object references across a gateway and methods of invoking an object across a gateway.

In a still further aspect, the present invention provides a gateway for being situated between a first network and a second network, comprising:

first interface means for receiving a message (q) including a request for some information relating to an object;

interceptor means for passing the request to a repository containing respective information about objects and receiving a reply from the repository; and

second interface means for returning a message (q') to the originator, including information contained in the reply.

This aspect of the invention relates to a way in which a gateway can publish the availability of objects across the gateway. Preferably, availability is reflected directly by whether an instance of the object is included in a repository or not. A particularly convenient repository in CORBA is the Naming Service. Of course, this system is also available from either side of the gateway.

Brief Description of the Drawings

Embodiments of the invention will now be described in more detail, by way of example only, with reference to the drawings, of which:

Figure 1 is a diagram which illustrates how an object reference is modified as it crosses a gateway from the inside to the outside;

Figure 2 is a diagram which illustrates how an object reference is modified as it crosses a gateway from the outside to the inside;

Figure 3 is a diagram which illustrates how a gateway can dynamically generate a proxy for an object;

Figure 4 is a diagram which illustrates how name tags can be used to control object availability;

Figure 5 is a diagram which illustrates how location tags, or identifiers, can be used to determine how an object reference should be mapped; and

Figure 6 is another diagram which illustrates how location tags, or identifiers, can be used to determine how an object reference should be mapped.

Annexes 1 and 2 which are attached hereto and are incorporated herein by this reference provide further detailed description on aspects and features of embodiments of the present invention. In particular, Annex 1 includes a description which relates in general to gateways, and specifically to a gateway configured for operation with objects. The description introduces the idea of a gateway that detects object references in order to make provision for a proxy, or interceptor, to handle invocations that are based on object references that cross the gateway. Annex 2, describes a detailed design for an Object Gateway using HP ORB Plus that will allow full CORBA interaction across firewalls implemented on a trusted operating system.

Best Mode For Carrying Out the Invention, & Industrial Applicability

Figure 1 is a diagram that illustrates the mechanism for mapping an object reference 125 which originates from a host 100 on the inside to the outside of a gateway 105, located on a host, hx. The object reference refers to an object 102, also located on the inside of the gateway. The gateway includes a proxy, or interceptor, for a message 120, which contains the object reference 125. In practice, the message 120 will commonly be an object invocation for a remote object (not shown), which includes an object key, an operation name, and respective arguments (the details of which are beyond the scope of the present description). The object reference 125 is included in the object key of the invocation.

As illustrated, the object reference 125 comprises (in the object key) an IOR[hi: pi: ki], where h, p and k represent host, port and object key values respectively for the referenced object 102.

The proxy 110, receives the message 120, extracts the object reference 125 therefrom and maps the object reference to form a modified object reference 135. The modified object reference 135 is contained in a forwarded message 130.

As shown, the modified object reference includes a new host name, hx, a respective port number, px, and an object key, Kio. A detailed description of the mapping function, mapio, used to generate the modified object reference, is provided in Annex 2. The object key, Kio, includes a number of features, as follows:

a name (or revocation) tag 140 which is unique to the proxy 110 which maps the object reference;

an identifier (or location tag) 145, which indicates from which side the object reference originated, which in this case is from the inside (i);

an IOR 150, which is the same as the original object reference 125; and

check data 155, which is a hash, or signature, of the name tag 140, the identifier 145, the IOR 150 and a secret S.

The reasons for incorporating the name tag 140, the identifier 145 and the check data 155 are explained in the introduction hereto, and in more detail in Annex 2. As such, they will not be discussed here as well.

When the message reaches the remote object, the nature of the information in the message indicates to the remote object that an object reference (the modified object reference 135) can be used to initiate an invocation of the object 102, by opening a connection with host, hx, on port, px, and sending an invocation including object key, Kio.

Figure 2 is a diagram that illustrates the mechanism for mapping an object reference 125 which originates from a host 100 on the outside to the inside of a gateway 105, located on a host, hx.

The mechanism is similar to the one in Figure 1, and corresponding features are labelled accordingly, except the labels in Figure 2 are of the form 2xx, rather than 1xx.

The significant difference is that the mapping function for this mechanism, mapoi, generates a modified object reference with an object key, koi, which includes an identifier 245, which specifies outside, o, rather than inside. Again, the mapping function, mapoi, is described in detail in Annex 2.

A detailed description of object invocations, which pass object references in both directions, is given in Annex 2.

Figure 3 is a diagram that illustrates the mechanism used by a gateway 305 for dynamically activating a proxy 315, or interceptor, as a result of receiving an object invocation 335 containing an object key 340.

The object key 340, includes a name tag 345, an identifier 350, which in this case is an "o" to indicate that the object reference originated on the outside of the gateway, an IOR to the destination host 300, and check data 360.

The invocation is received on a port, py, of the host, hy. The particular host, port and object key combination were defined in the modified object reference, as explained with reference to Figure 2.

The gateway 305 is configured firstly to attempt to locate a proxy for the object referenced in the object key of an object invocation received on the port py. If no proxy is located, the gateway is configured then to pass the object invocation received on port py to an activator 320, to dynamically generate the appropriate proxy 315. Dynamic proxy, or interceptor, generation is described in more detail in Annex 2.

Once the proxy 320 has been generated, it processes the invocation, as will be described in more detail in relation to Figures 5 and 6, an forwards an invocation to the destination object 300.

Figure 4 is a diagram that illustrates a mechanism for locating, or resolving, object references 420 across a gateway 400.

The diagram shows an incoming message 401 which is a request to resolve an object name, using a Naming Service 410 in CORBA. In effect, the Naming Service 410 returns an object reference in response to an object name. A similar mechanism can be used by other naming services, traders or object locators in a CORBA system or in other systems.

The message 401 is received by a Naming Service proxy 405, which is configured to forward all resolve requests in messages 402 to an external context 415 defined in the Naming Service 410. The external context 415 is an access point in the Naming Service for all requests from the outside of the gateway 400.

Any objects that are available from outside the gateway 400 have an entry 420 in the external context and can thus be resolved. The result of a successful resolve request is the return by the Naming Service 410 to the proxy 405 of a message 430, including an object reference 435 for an appropriate object inside the gateway 400. The proxy 405 forwards a respective message 440 to the requestor. The Naming Service 410 generates an exception for unsuccessful resolve requests, resulting from there being no respective object entry below the external context 415. Again, the proxy forwards the exception to the requestor.

In forwarding the message 430 for a successfully resolved request, the proxy 405 extracts the object reference and generates a modified object reference 445. The modified object reference includes specific host and port details for the gateway and an object key comprising a name tag, an identifier, an IOR and check data, as explained above with reference to Figure 1.

Figures 5 and 6 are diagrams which illustrate two examples of the use of identifiers, or location tags.

Figure 5 illustrates the case where an object reference 515 is passed in a message 510 from the outside of a gateway 500 to the inside. The gateway 500, using the mapoi function, modifies the object reference 515 to form a modified object reference 525, which is forwarded on in a message 520.

The modified object reference 525 includes, among other things, an identifier 526 and check data 527.

If a message 530, including the modified object reference 535, is received by the gateway 500, from the inside thereof, the gateway uses the mapio function on the check data 527 to verify the contents of the object key, koi, by repeating the hash operation.

In the example shown, the gateway 500 will be able to verify the contents of the modified object key. Further, the gateway 500 recognises that the object reference originated from the outside of the gateway 500. Thus, the gateway 500 'unmodifies' the modified object reference 535, rather than modifying it again, by simply extracting the original object reference 515, and forwarding it as an object reference 545 in a message 540.

Figure 6 illustrates the case where an object reference has previously been modified by crossing from the outside to the inside of a gateway 600, to form a modified object reference 615. Somehow, for example by an out of band method such as email, the modified object reference 615 ended up back outside the gateway 600. When the modified object reference 615 is passed in a message 610 from the outside of the gateway 600 again, the gateway knows, from the identifier 627, that the object reference originated from the outside. As a result, the gateway 600 knows that the object reference 615 does not need to be modified again, and the reference simply passes through without being mapped.

Some of the advantages of the present invention will now be considered.

In accordance with embodiments of the present invention, gateways can be daisy-chained. In other words, object reference can cross multiple gateways, even when a gateway has no knowledge of the object that is referenced, and an invocation of the referenced object

can pass back through the gateways. This is a result of the present invention being transparent to users and system administrators alike.

The detection and replacement of object references embedded in messages means that there is no need for end to end encryption or authentication.

The gateways appear to be the real "clients" and "servers" to those accessing them.

Other features and advantages of the present invention will become apparent to those skilled in the art of distributed systems on reading the present description in combination with Annexes 1 and 2.

REFERENCES

[NCSC]: National Computer Security Centre, "Department of Defence Trusted Computer System Evaluation Criteria", DoD Standard 5200.28-STD, 1985

[DIA 91]: "Compartmented Mode Workstation Evaluation Criteria VERSION 1 (Final)", J.P.L. Woodward, DDS-2600-6243-91, 1991.

CMW Manuals

HP-UX Trusted OS Installation Manual

HP-UX Trusted OS Read Me First / Release Notes

HP-UX 10.09.01 CMW Trusted Facilities Manual

HP-UX 10.09.01 CMW Key Security Concepts

HP-UX 10.09.01 CMW Support Media User's Guide

HP-UX 10.09.01 CMW Trusted Facility Admin Ref. Manual

HP-UX 10.09.01 CMW MaxSix Administrator's Guide

HP-UX 10.09.01 CMW Security Features User's Guide

HP-UX 10.09.01 CMW Security Features Programmer's Guide

ANNEX 1

INTRODUCTION

The following description relates in general to gateways, and specifically to a gateway configured for operation with objects. The description introduces the idea of a gateway that detects object references in order to make provision for a proxy, or interceptor, to handle invocation that are based on object references that cross the gateway.

The description first illustrates how a simple gateway can invoke proxies whenever a reference is passed across the gateway, and then develops this concept by suggesting that proxies should be generated dynamically only when required, to minimise state build-up. Annex B describes in much more detail the implementation details of these aspects.

The description also describes in detail how the CORBA Naming Server can be used very conveniently to control how objects are published or not across the gateway. The Naming Server returns an object reference, where one is present, in response to receipt of a object name. This service can be likened to a telephone 'white pages' directory.

The gateway can be deployed on conventional operating systems such as HPUX. However, it is also implemented on a trusted platform which is B1 compliant with some B2 and B3 features. This provides extremely high assurance in the security of the gateway.

The gateway uses only standard features of CORBA and is completely transparent to both the client and server ORBs (Object Request Brokers). This means that any CORBA compliant ORB can be used to implement client and server objects which communicate via the gateway. This transparency also means that the gateway is not needed during application development and is needed only when the application is deployed across firewalls.

This document describes the purpose, function and features of the gateway, by describing the problem being solved and how the gateway provides a solution to this problem through a number of use scenarios. Finally a number of features are described, the most important is the

use of trusted operating system technology to provide extremely high assurance in the solution.

1 A Simple Gateway Solution

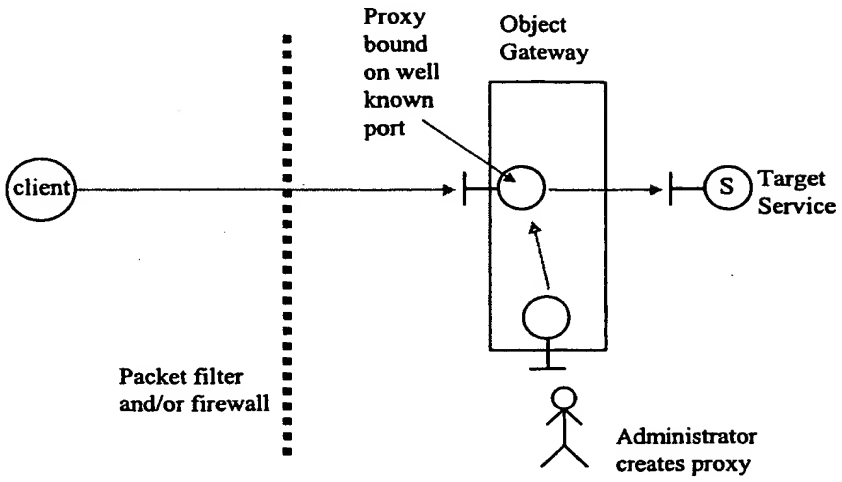


Figure 1: Making a CORBA object available to an external client

At its simplest, the problem we want to solve is to provide a gateway which controls the visibility of internal CORBA (IIOP) services to clients outside the firewall. The gateway is located on a well known host and listens on a well known port, so that the firewall or packet filter is able to block attempts to connect to other ports on that host. An administrator wishing to make a service, S, available to external clients creates a proxy object for S. This is shown in Figure 1.

The proxy forwards incoming invocations to S; it runs inside the gateway and shares the same port as other proxies in the gateway. Before an invocation is forwarded to S, the proxy may perform certain checks such as checking that the invocation is type-safe, checking that the parameters are in range and checking that the client is authorized to make the invocation. A proxy could choose to check the results of the invocation of S, before forwarding them back to the client.

The client is given an object reference to the proxy, rather than an object reference to S. Any attempt by the client to use an object reference to S will fail: blocked by the firewall, because S will be on a host and port number not allowed to receive incoming packets.

One very powerful feature of CORBA is the ability to hand object references between clients and servers as parameters and results. A server can create an object and include a reference to it in the result that it returns to the client. A very well know example is the CORBA Naming Service "list" operation which returns a reference to an iterator object in its reply to the client [OMG 97a]. This object reference needs to refer to a proxy running in the gateway, rather than iterator object itself which is not available directly through the firewall. This situation is show in Figure 2.

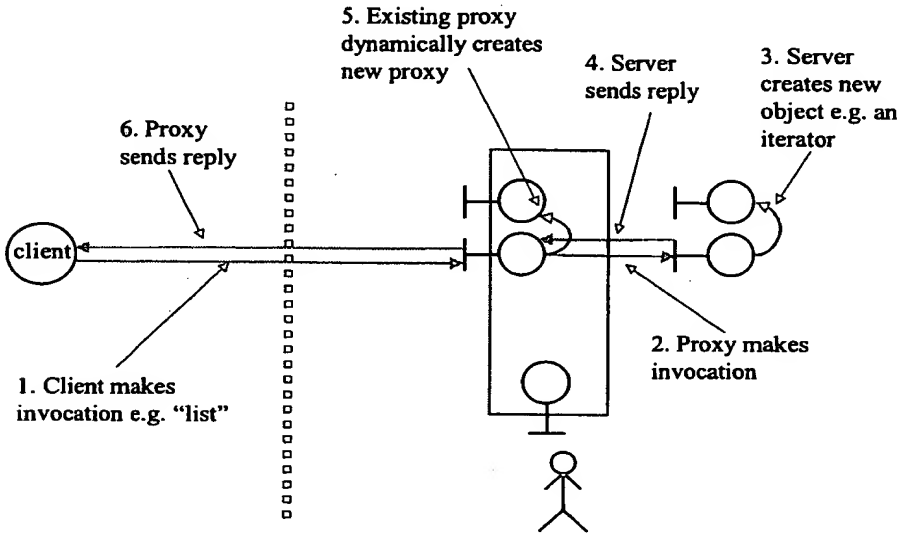


Figure 2: A server returning an object reference to a client

When the existing proxy receives a reply containing an object reference, it needs to arrange that the object reference is changed so the object gateway will intercept calls made using the new object reference. Conceptually, one can think of the gateway as detecting the object reference, and replacing it with a reference to a dynamically created proxy, as shown in Figure 2. However, this will lead to a garbage collection problem: the client may never use the object reference, or the service to which the object reference refers may be destroyed (e.g. when the client invokes the iterator's destroy() operation). In general the gateway, will have no way to tell when the new proxy is no longer needed.

Figure 3 shows the equivalent problem which arises when a server receives a reference to a callback object from a client. (This paradigm is used in the CORBA Event service [OMG 97a].) The server is unlikely to be allowed to send packets through the firewall to invoke the callback object directly or to be able to receive the packets containing the reply, so the gateway must detect the incoming object reference and replace it with a dynamically created

proxy. Again there is a garbage collection problem: how does the gateway know when the callback proxy is no longer needed?

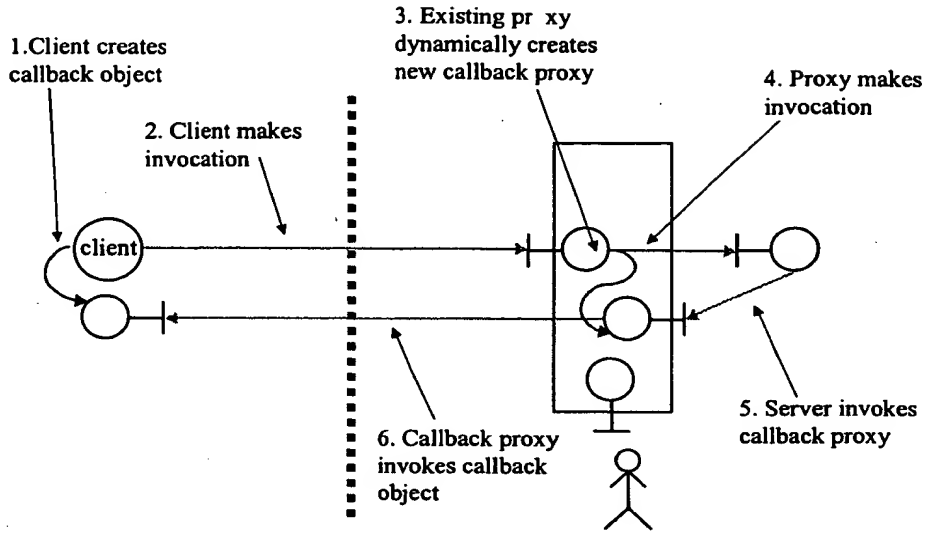


Figure 3: A client passing an object reference to a server

So far our discussion has assumed that the client is not also behind a firewall. Unfortunately in most enterprises it will be. Thus any object gateway needs to provide both a client-side and server-side solution. Figure 4 illustrates the basic requirements to allow a client and server interaction in which object references are exchanged freely. The steps which occur are as follows.

1. The client creates a callback object.
2. The client makes an invocation request containing the reference to the callback object. This call is intercepted transparently by the client gateway proxy.
3. The proxy detects the object reference for the callback object and creates a proxy for it.
4. The client proxy makes an invocation request replacing the object reference with one for the proxy created in step 3. The call is intercepted transparently by the server gateway proxy.
5. The server gateway proxy detects the object reference in the request and creates a proxy for it.
6. The server gateway proxy sends an invocation request replacing the object reference with one for the proxy created in step 5.
7. The server creates an object.
8. The server sends a reply containing a reference to the object created in step 7.
9. The server proxy detects the object reference and creates a proxy for it.

10. The server proxy sends a reply containing an object reference to the proxy created in step 9.
11. The client proxy detects the object reference and creates a new proxy for it.
12. The client proxy sends a reply containing a reference to the proxy created in step 11.

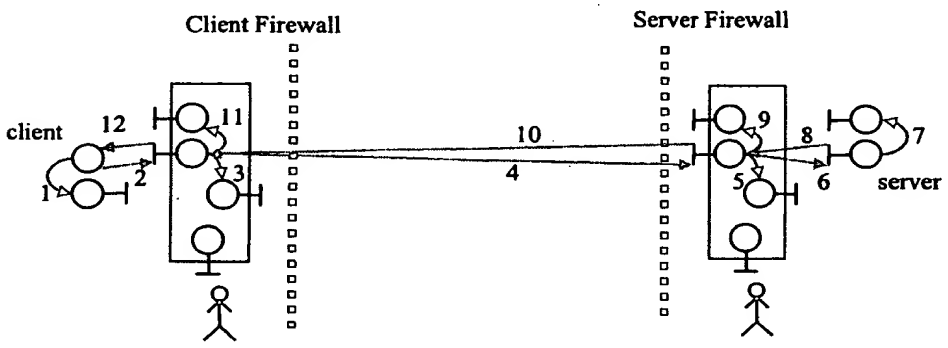


Figure 4: client-side and server-side gateways

Simple implementations of the above functionality would lead to the build up of a tremendous amount of state in busy object gateways and subsequent garbage collection problems. The gateway uses algorithms which avoid this.

So far we have described a single client attempting to invoke a single server. The reality is that there will be many different clients trying to use many different servers, as shown in Figure 5.

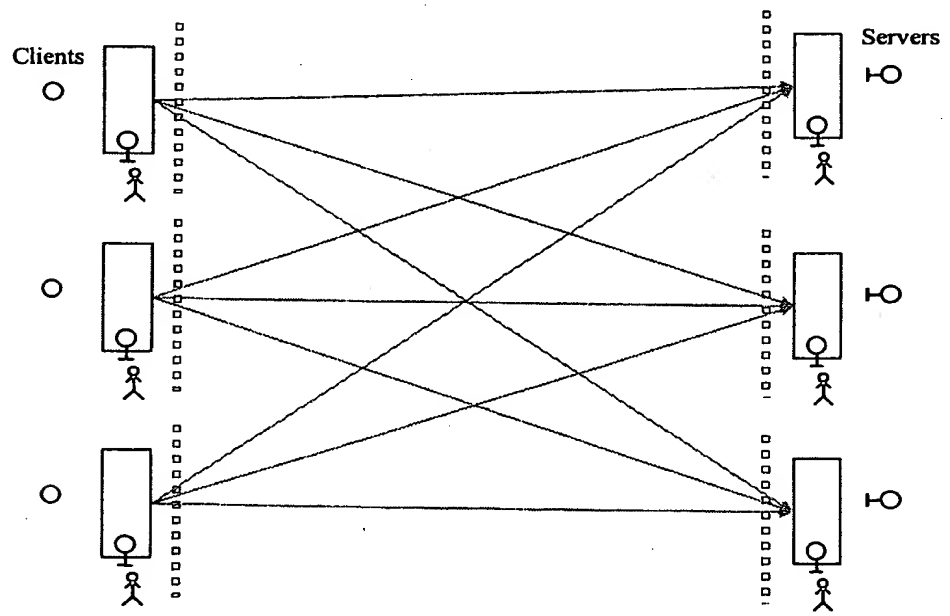


Figure 5: The need for collaboration between gateway administrators

It may be reasonable to expect server gateway administrators explicitly to enable access to each service being made available from their internal network. This is not a reasonable requirement to impose on the administrators of all client gateways: some enterprises may want to allow their users to invoke any service they choose. If this is the policy, it is not reasonable to expect client administrators to anticipate and explicitly enable access to each service their clients might want to use; they will need some way to make their gateways promiscuous. Other client administrators may want to allow access to any service made available by certain business partners, but to exercise tight control over anything else.

2 Preferred Gateway Solution

Figure 6 shows howthe gateway acts as a server-side gateway providing controlled access to internal services. The administrator configures the gateway to act as a proxy for a particular context in a Naming Server [OMG 97a] or a trader lookup interface [OMG 97a] — we call the selected interface "the bootstrap point". For the remainder of this text we shall assume a Naming Server Context is used, but the same principles apply to the use of trading technology.

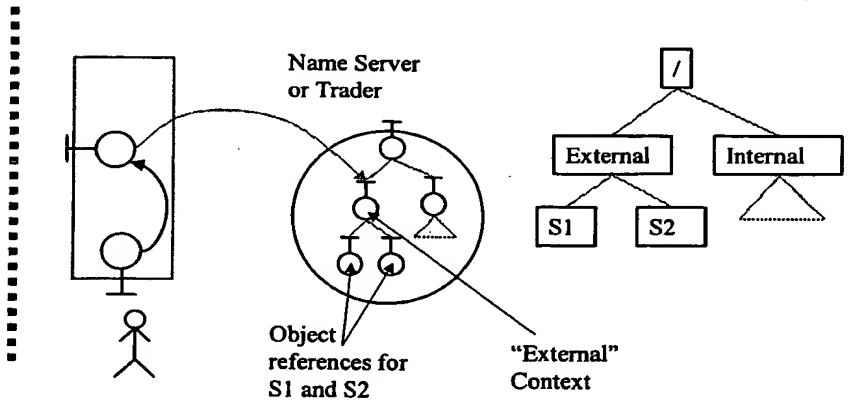


Figure 6: Simple operation of server-side gateway

Once the gateway is told to act as a proxy for a naming context, that context and everything it contains becomes visible to external clients. Clients can browse the naming context and retrieve objects (references) from it. The gateway will act as a transparent proxy for any object, O, which is retrieved from the Naming Context. Furthermore, it will act as a transparent proxy for any object which is retrieved by invoking O, or any object handed by the client as a parameter to O. This means that the administrator must be confident that the

services which are available in the "External" naming context do not introduce trap doors by handing out object references to services which should not be made available to external clients. Extra filtering technology can be used to ensure the gateway can only act as a proxy for services on certain hosts.

If a service is removed the naming context the gateway will cease to act as a proxy for that service and any object references obtained from that service.

Conceptually each time the gateway sees a request or a reply with an object reference, it creates a proxy for the object and replaces the object reference with one for the proxy. The administrator bootstraps the process by choosing the bootstrap point and creating the bootstrap proxy.

Choosing the bootstrap point controls what is available through the gateway. The gateway's default behaviour is to proxy all objects obtained directly or indirectly from a bootstrap proxy object. An object is obtained indirectly from a bootstrap proxy object if its reference was obtained from a third party whose object reference was itself obtained directly or indirectly from the bootstrap proxy object. The gateway also acts as a proxy for all objects passed as parameters in requests to objects for which it is acting as proxy (i.e. callback objects). The remaining examples in this section demonstrate the power of this paradigm for both client and server gateways.

In thethe gateway implementation proxies are not created each time an object reference passes through the gateway. The algorithms minimize the state which the gateway needs to hold, so that many object references can pass through the gateway without internal state changes occuring. This improves the efficiency of the solution.

By defaultthe gateway proxies are implemented using the Dynamic Invocation Interface and Dynamic Skeleton Interfaces defined in [OMG 95]. This ensures that all messages relayed by the gateway are correct IIOP messages and that both requests and replies are type-safe. They do not guarantee the application semantics will not be violated (e.g. operations called in the wrong order). Application specific proxies can be provided to support this (see section 3.5).

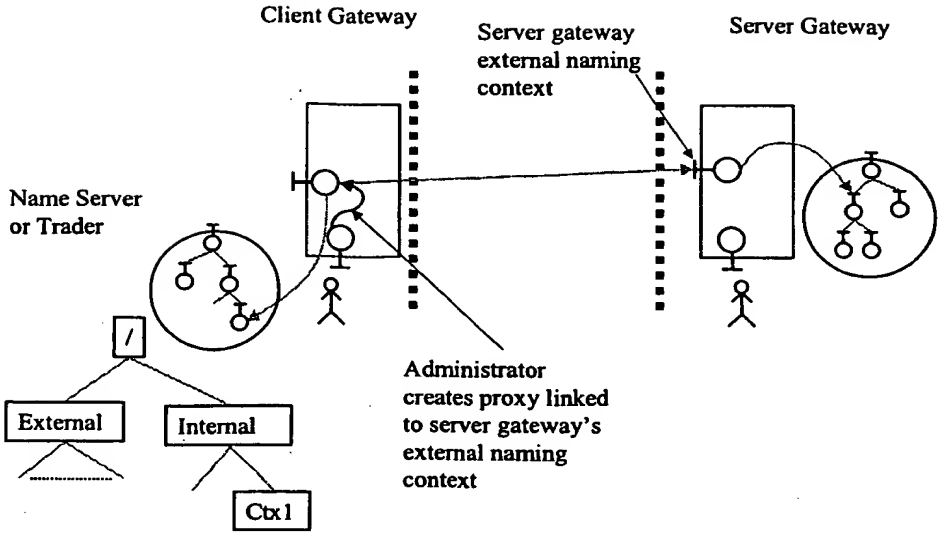


Figure 7: Client and server gateways interoperating

Figure 7 shows how a client gateway interoperates with a server gateway. In this case, we have assumed that the client administrator wants to make available to his or her users all services available from the server gateway. This is done by creating a bootstrap proxy for the Server gateway's external naming context and placing the object reference for that proxy in a Name Service or trader so that it is accessible to clients. This will cause the client gateway transparently to act as a proxy for the server gateway external context, or any object O retrieved from this context. Furthermore, it will act as a transparent proxy for any object which is retrieved by invoking O, or any object handed by the client as a parameter to O. This means that the administrator must be confident that the clients do not introduce trap doors by handing out object references to services which should not be made available externally. Extra filtering technology can be used to allow only proxies for services and callback objects on certain hosts, but this may not be an issue if the administrator has configured the object gateway so that it interacts only with the gateways of "trusted" business partners.

Although not shown in Figure 7, the client gateway could also act in the role of a server gateway, making services available to external clients. Similarly the server gateway could also act as a client gateway for its internal clients. The two different parts of the gateway would have separate contexts in the Name Server ("Internal" and "External").

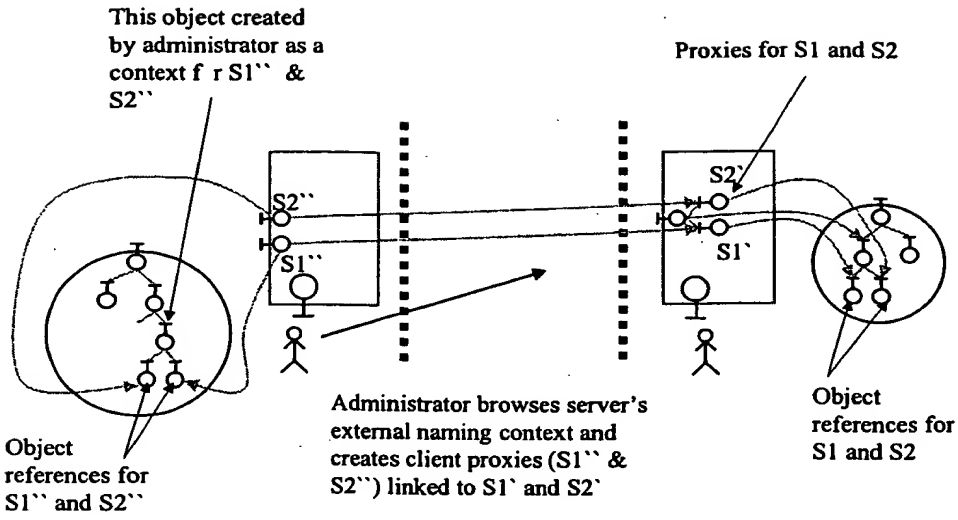


Figure 8: Making selected services from a site available to users

Some client administrators may not want to make all the services available from a particular site. This situation is shown in Figure 8. Here the client administrator wants to allow access to S1 and S2 only. So these are chosen as the bootstrap points, rather than the external Naming Context. First the administrator creates a context in the local naming server. Next he or she browses the external naming service to retrieve the object references for the relevant proxy objects S1` and S2` and creates client side bootstrap proxies for these (S1`` and S2``). Finally the object references for S1`` and S2`` are inserted into the local naming service under the appropriate context. The administrator's users can now retrieve object references from their local naming service which will allow them to access S1 and S2, but no other services.

Some administrators may want to allow their clients to access any service they choose (promiscuous mode). The simplest way of doing this would be to create a bootstrap proxy for some public directory (e.g. trader or naming service) in which server administrators or some other party had already registered their gateway's external naming service or trading service. Then the gateway would act as a proxy for any service S retrieved from the public directory and in turn any service whose object reference was obtained as a result of accessing S. Thus choosing a public "directory" service as the bootstrap point makes the gateway promiscuous. This is shown in Figure 9.

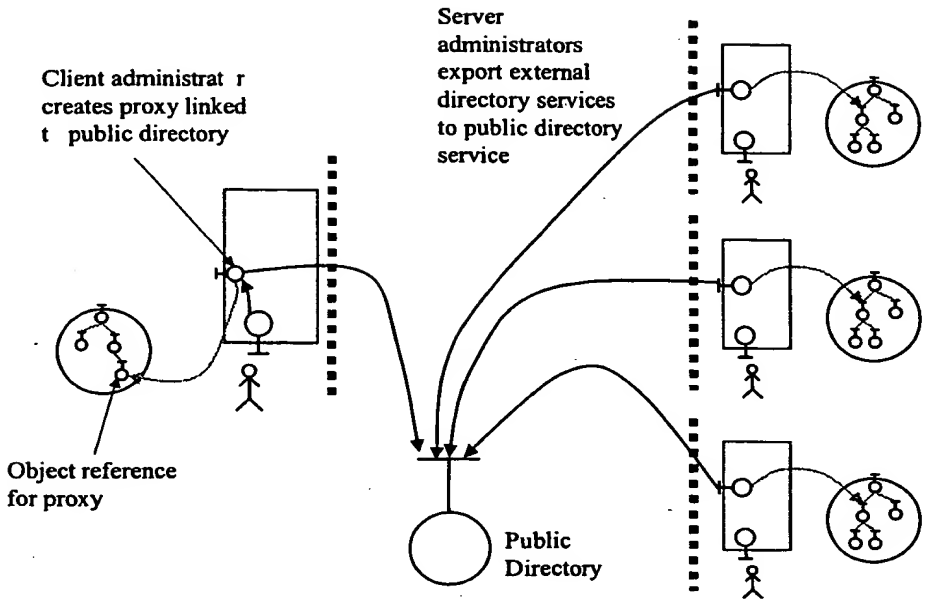


Figure 9: Putting a gateway into promiscuous mode using a public directory

The gateway is entirely transparent to clients and servers. It is also transparent to other instances ofthe gateway (unless our secure gateway to gateway transport is used — see section 3.2). This means that gateways can be daisy chained, if a request needs to traverse multiple firewalls. This will be necessary if an enterprise uses firewalls to partition its network internally and different firewalls to restrict access to the Internet.

The current version ofthe gateway is implemented using HP ORB Plus and listens on three well known ports, unless application specific proxies are used (see section 3.5).

3the gateway Features

3.1 Trusted operating system technology — the foundation of secure gateways

Security requires the integration of secure communications (authentication, confidentiality, etc) and secure platforms — a trusted operating system. Secure communications makes it difficult for an attacker to cause harm by intercepting a message as it passes through the network. A secure platform makes it difficult for an attacker to cause harm by gaining unauthorized access to data stored on that platform (on a disc, in a process etc). As well as running on conventional operating systems such as HPUX,the gateway also runs on two trusted operating systems: HPUX CMW and the VirtualVault Operating System (VVOS).

The HPUX CMW operating system is an implementation of the Compartmented Mode Workstation, this system was designed to meet military security requirements and offers much more security than can be provided by conventional operating systems such as Windows NT or standard variants of Unix[1]. CMW was designed to prevent information leakage or the disclosure of information to unauthorized users — including those attacking the system.

The facilities provided by CMW to prevent information leakage are ideal for building a gateway to provide controlled access to information and services in Internet and Intranet based environments. VVOS is a derivative of HPUX CMW optimized for gateway installations. The best known example of its use is the HP VirtualVault, which provides a highly secure WWW server. VirtualVault has been used successfully in a number of Internet banking applications, where security is vital. A typical VirtualVault configuration is shown Figure 10. Here the WWW server is used to provide controlled access to internal data (e.g. bank accounts). Internally the VirtualVault machine is configured into two compartments. These compartments are used to separate critical components so that they cannot interfere with each other (information leakage). Any information that does flow between compartments is audited. Thus the web server that provides the external interface and the programs used to access internal data are in separate compartments. The only communication path between the two compartments (and hence information flow) is via the special trusted (privileged) process. The latter is responsible for auditing and controlling the information flow.

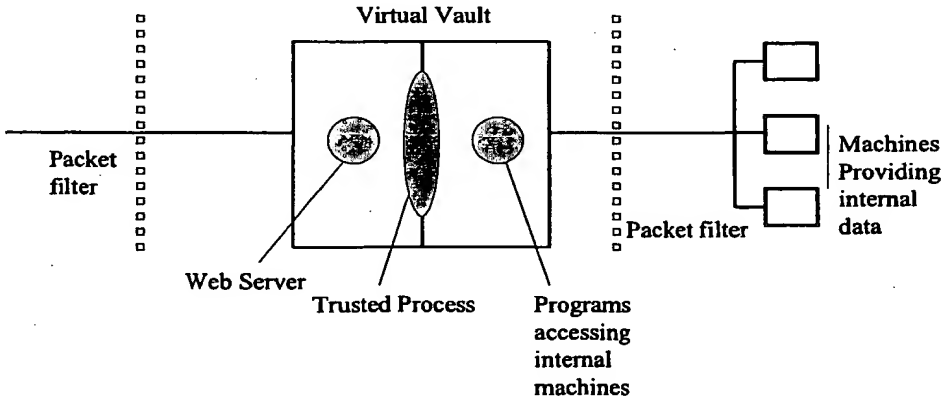


Figure 10: A typical VirtualVault installation

[1] To be specific, HPUX CMW is B1 certified and has some B2 and B3 features, Windows NT and ordinary Unix are only rated at C2 (at best). We know of no general purpose platforms with better than a "B" rating.

Note that the installation still makes use of firewall technology such as packet filters. VirtualVault is complementary to conventional firewall technology, not an alternative. Its value is that the gateway machine visible through the firewall is much more robust to attack than a conventional Unix or Windows NT machine.

Although a typical VirtualVault installation has two compartments ("Inside" and "Outside"), both CMW and VVOS can support more compartments if needed. Each compartment can used to be isolate different system components.

3.2 Secure transports

The current implementation of the gateway uses standard CORBA IIOP (as defined in [OMG 95]). In addition we are implementing gateway to gateway SSL based transport in which the IIOP message is tunneled over SSL. We also plan to support the standard IIOP over SSL mapping when that becomes available.

One consequence of the strategy of object reference replacement, is that confidentiality and integrity become point-to-point issues. This is because each time a message is passed through a gateway that gateway may need to change it. This leads to a chain of trust involving the client, the server and all the gateways between them. It does not necessarily cause extra programming complexity as gateways are transparent: they look like "real" clients to a server and "real" servers to a client.

3.3 Authorization and restricting operations

We are implementing a facility to allow administrators to restrict the availability of certain operations in certain services. For example, a server gateway administrator may want to think hard about allowing certain operations defined in the Naming Service to be invoked through the gateway. This could allow an external client to update the list of external services.

It may make sense to allow authenticated clients to update some context within the Name Service. Consider the case when the client is a trusted business partner. By allowing them to update a context in your naming service they can make new services available to your users and ensure the gateways will act as proxies for the new services. Figure 11 shows what

happens conceptually. In practice no state change will occur in either gateway at the time of the bind operation, so no separate proxy objects will be created.

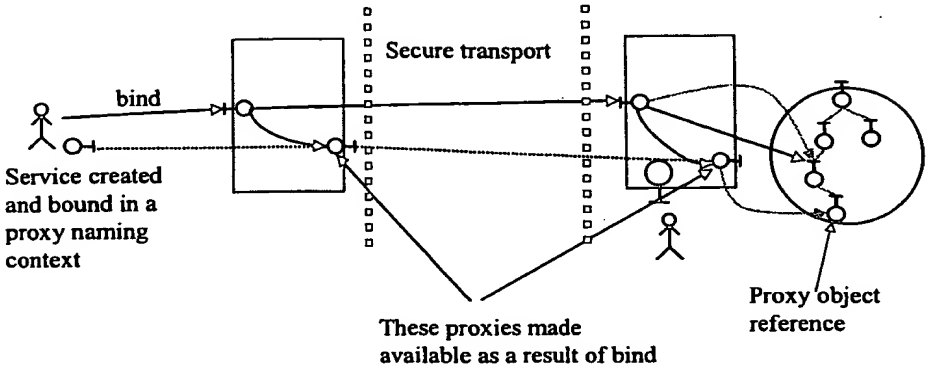


Figure 11: Allowing selected external updates of a naming service context

3.4 Visualization and the management interface

The gateway has an IDL [OMG 95] defined management interface (not available to external clients). This allows an administrator to change the configuration of the gateway. Example administrative operations include: changing what operations are available on an object to external clients and withdrawing the rights to access external services from internal users. The management interface also allows the administrator to see what services have been made available to external users and internal clients.

At the time of writing a Java based GUI and an HTML forms (WWW) user interface are under development for this management interface. These user interfaces give a visual representation of the state of the gateway.

3.5the gateway API

The gateway provides an API so that application specific proxies can be incorporated into the gateway. Application specific proxies can use IDL generated stubs and skeletons and allow programmers to embed knowledge of the applications correct semantics (a simple example is restricting the range of parameters). The current implementation requires the gateway to be recompiled to incorporate an application specific proxy, the next version will require no gateway recompilation or restarts to install an application specific proxy. Application specific proxies are run in a separate process, so a version ofthe gateway using these will require four

or more ports to be visible through the firewall (at least one extra port for each additional process needed).

4 Summary

The gateway is a security gateway for CORBA. It is transparent to both client and servers and uses only standard CORBA features. Hence it can be used with any CORBA compliant ORB and is not needed until the application is deployed across firewalls. It has been implemented on a trusted operating system (as well as a conventional platform), which provides extremely high assurance in the security of the solution.

The gateway is compatible with existing firewall machinery: it runs on a single host and listens on a set of well known ports (3 by default in the current implementation).

The gateway is both a client side and a server side gateway: administrators can control what external services their users can access and what services are made available to external clients. By carefully choosing the objects which are the "bootstrap points", the administrator can exercise very fine control, or make the gateway fully promiscuous.

The gateway's object reference detection and replacement machinery means that gateways can be daisy chained to support interactions across multiple (more that two) firewalls. The algorithms used minimize the need for gateway state and avoid a garbage collection problem which could arise if many objects are created dynamically.

In the futurethe gateway will allow administrators to control which operations are available on a particular object. The management interface (under development) provides administrators with visual feedback on the state of the gateway as configuration changes are made.

The gateway provides a secure transport for secure gateway to gateway communication (under development). In the future it will also support IIOP over SSL.

By default,the gateway proxies guarantee that all requests and replies are type-safe and are correct IIOP invocations. Thethe gateway API means that application specific proxies can be

provided which have embedded detailed semantic knowledge of the interaction which should be taking place. This allows such proxies to exercise very detailed control.

References

[OMG 97a] CORBAservices: Common Object Services Specification, OMG, July 1997

[OMG 97b] The Common Object Request Broker: Architecture and Specification, Revision 2.1, OMG, August 1997

[OMG 95] The Common Object Request Broker: Architecture and Specification, Revision 2.0, OMG, July 1995

ANEX 2

Introduction

This document describes a design for an Object Gateway using HP ORB Plus that will allow full CORBA interaction across firewalls implemented on a Trusted OS (CMW or VVOS). The design allows the interactions to be controlled but does not require continuous human intervention.

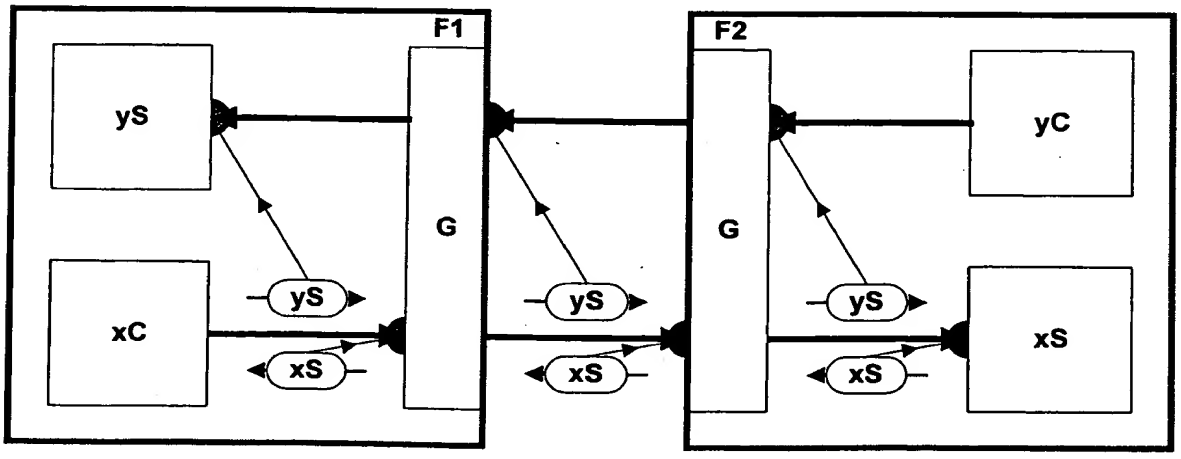
Although the description specified ORB Plus in places, the design can be adapted to any other ORB that provides similar functionality.

The gateway can be implemented on a conventional operating system if this is considered to provide acceptable security. In this case, either a simplified form of the trusted relay may be used, or the functions performed by the trusted relay for the interceptors may be incorporated into the interceptors themselves.

CORBA Firewall Scenario

CORBA is becoming well established as an Enterprise Distributed Computing infrastructure. In order to move on to using CORBA in Extranet applications, it is necessary to provide a way to allow CORBA objects to interact across multiple firewalls. Figure 1 shows the basic scenario, see "Nomenclature" on page 32 for an explanation of the conventions used in this and the following diagrams.

Figure 1 CORBA Invocations across Multiple Firewalls



A client xC inside one firewall, F1, wishes to invoke a server xS inside a different firewall, F2. This means that the invocation must pass out through the object gateway G in one firewall and in through the object gateway in another. If there is an object reference in the request, e.g. the reference to yS in the diagram, this must be transformed so that it can be used in the destination environment. Similarly, a reference passed in a response, e.g. a reference to xS in the diagram, must be transformed to be usable in its destination environment.

The CORBA location transparency mechanisms, which allow servers to be restarted on different hosts, and to listen on different ports, must also work properly across the firewalls.

The object gateway described in this document provides the reference transformation and connection establishment required for transparent access across the firewalls. It also incorporates a mechanism to control or revoke access to servers, even where the references were passed as parameters.

The object gateway is designed to be deployed on a host with multiple network interfaces running a Trusted Operating System supporting Multi-Level Security. The gateway is designed for a configuration where the sensitivity labels are set up to prevent unprivileged process from having access to more that one network. The object gateway may also be deployed on conventional hosts; typically, this would be on a bastion host that is part of a firewall.

The object gateway components are described in terms of an implementation on HP ORB Plus. A description at this level is necessary in order to understand how the components and interactions that are not normally visible to an application programmer work across the gateway.

CORBA Interoperability Features

The object gateway design exploits the interoperability facilities that are in the CORBA specification [OMG 95]. In particular, it uses the features of Interoperable Object References (IORs), and the Internet Inter-ORB Protocol (IIOP). IORs and IIOP are described briefly here.

Interoperable Object References

Interoperable Object References are defined in the CORBA Specification using IDL shown in Figure 2.

Figure 2: IOR definition from CORBA IOP module

```
module IOP{                                          // IDL
    //
    // Standard Protocol Profile tag values
    //
    typedef unsigned long ProfileId;
    const ProfileId TAG_INTERNET_IOP = 0;
    const ProfileId TAG_MULTIPLE_COMPONENTS = 1;

    struct TaggedProfile {
        ProfileId tag;
        sequence <octet> profile_data;
    };
    //
    // an Interoperable Object Reference is a sequence of
    // object-specific protocol profiles, plus a type ID.
    //
    struct IOR {
        string type_id;
        sequence <TaggedProfile> profiles;
    };
// Remainder of module deleted.
};
```

For the description of the generic gateway components, the important feature is the sequence of tagged profiles. The type_id will be used later when describing how to specialise some of the components.

Each tagged profile consists of a profile identifier (e.g. "TAG_INTERNET_IOP") and the profile data. In the simplest IOR, for a service supporting the IIOP protocol, the profile sequence will only have a single TaggedProfile data item having the tag "TAG_INTERNET_IOP". This indicates that the sequence contained in the profile_data field

is a ProfileBody as defined in the IIOP module. Figure 3 shows the IIOP version 1.0 definition from [OMG 95].

Figure 3: IIOP profile from the CORBA IIOP Module

```
module IIOP {                                    // IDL
    struct Version {
        char major;
        char minor;
    };
    struct ProfileBody {
        Version iiop_version;
        string host;
        unsigned short port;
        sequence <octet>object_key;
    };
};
```

The ProfileBody consists of the IIOP version number, host name or IP address, TCP/IP port and an opaque sequence of octet (bytes) known as an object key. The host address and port specify where the server is to connect; the object key is opaque and is sent over the connections as part of an IIOP request by the client. The server process uses this to identify the object when it receives an invocation. Since the client is not allowed to interpret the object key, (it is just an opaque bag of bits) we can use this to store whatever information we like. We use it to store the object reference of the target service accessible through the gateway.

In the descriptions of the gateway, the host, port and object key are the significant data in an IOR profile. IORs will be shown as IOR[h:p:k] for an IOR with a single profile, so that the relevant items can be picked out in the descriptions. An IOR with two profiles will be written IOR[h1:p1:k1,h2:p2:k2].

IIOP version 1.1 [OMG 97] adds a "components" field to the ProfileBody. This field provides extra information about the server for use by the client; e.g. the type of ORB, and which character sets are supported. This does not introduce any new issues for the gateway itself. The gateway relies on the underlying ORB to linearize (e.g. stringify) the object reference so

that it can be embedded as part of the object key in a new reference. This means that the ORB must linearize the additional information, and generate the new fields when extracting IORs.

Internet Inter-ORB Protocol

For simplicity, the descriptions will be in terms of only some of the features of IIOP, but the relevant components will be built using full-function ORBs supporting the whole protocol. IIOP is the General Inter-ORB Protocol transmitted over TCP/IP; the messages transmitted are GIOP messages

A GIOP request message is used to invoke an operation on some object; the response is a reply message. These are the only message types that will be discussed in the detailed design. The relevant part of GIOP, specified in IDL and taken from [OMG 95] is shown in Figure 4.

Figure 4: GIOP message formats from the CORBA GIOP Module

```
module GIOP { // IDL
    enum MsgType {
        Request, Reply, CancelRequest,
        LocateRequest, LocateReply,
        CloseConnection, MessageError
    };
    struct MessageHeader {
        char magic [4];
        Version GIOP_version;
        boolean byte_order;
        octet message_type;
        unsigned long message_size;
    };
    struct RequestHeader {
        IOP::ServiceContextList service_context;
        unsigned long request_id;
        boolean response_expected;
        sequence <octet> object_key;
        string operation;
```

```
        Principal requesting_principal;
    };
    enum ReplyStatusType {
        NO_EXCEPTION,
        USER_EXCEPTION,
        SYSTEM_EXCEPTION,
        LOCATION_FORWARD
    };
    struct ReplyHeader {
        IOP::ServiceContextList service_context;
        unsigned long request_id;
        ReplyStatusType reply_status;
    };
// Remainder of module deleted.
}
```

GIOP 1.1 [OMG 97] adds support for fragmentation. This does not introduce any new issues for the gateway. Although it is necessary to understand which messages are sent where, the gateway does not deal with the messages directly. The gateway interacts with the ORB at the language mapping level.

A request message consists of a GIOP message header, a Request Header, and a Request Body. The Request Body contains the in and inout arguments. The significant elements of the request are the object key, the operation name, and the arguments. In the descriptions below, request messages will be shown as k:op:args to allow these elements to be discussed.

A reply message consists of a GIOP message header, a Reply Header and a Reply Body. The content of the Reply Body depends on the status in the header. Replies with status LOCATION_FORWARD have an IOR as the body, these are considered explicitly in the description. Replies with status NO_EXCEPTION have the return value, inout and out parameters as the body. Other replies have the exception as the body. These cases are shown as status:results in the description as the difference is not significant for the behaviour of the gateway.

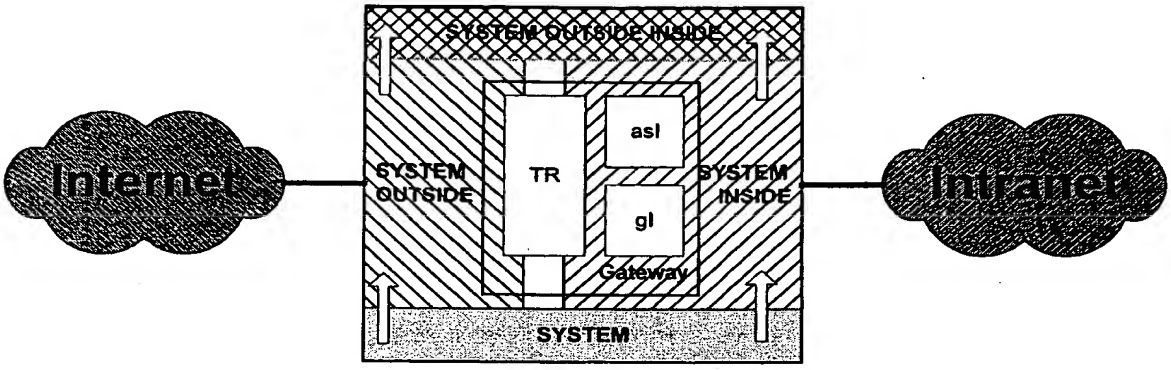
Although the descriptions will explicitly consider replies with status LOCATION_FORWARD, the client ORB is responsible for processing the reply and sending the request to the new destination. The important issues for the gateway design are identifying where the ORB needs to be modified, and arranging the use of IORs to confine such changes to the ORB supporting gateway components. It is important that the ORBs supporting the ultimate clients and servers do not need to be modified.

Gateway Structure

The overall structure of the gateway is determined by the need to run it on a trusted operating system that supports Mandatory Access Control and Multi-Level Security, and which supports privileges that allow fine-grained control over sensitive system facilities. In particular, the gateway is designed to run on the HP VirtualVault configuration.

The overall structure of the gateway is shown in Figure 5.

Figure 5 Gateway structure on VirtualVault



On VirtualVault, every resource is labelled with one of the four labels shown in the diagram, and the system enforces Mandatory Access Control rules based on these labels. Of particular interest here are the two network interfaces; one is labelled "SYSTEM OUTSIDE" and the other is labelled "SYSTEM INSIDE". The mandatory access control rules prohibit any access between resources with these labels, and in particular, any normal process that can connect to one is prohibited from connecting to the other.

The system has privileges that can be used to override the Mandatory Access Control, but it is important to minimise the components that have privileges.

The object gateway is structured into interceptors that do most of the work; these are shown in the diagram as 'gI' - a generic interceptor - and 'asI' - an application specific interceptor. These are relatively large and complex, and so it is not appropriate to give them privileges. The objective is to enable invocations from the outside (the Internet in the diagram) to reach inside services via the interceptors. Therefore there is a Trusted Relay shown as TR in the diagram that has the privileges required to override the Mandatory Access Control rules. It is the job of the trusted relay to ensure that all incoming invocations go to an interceptor. The interceptor will then perform the access control functions that defend the inside objects from rogue invocations.

Although the gateway can be implemented on a conventional operating system (with reduced security assurance), the descriptions that follow will include the trusted relay since this introduces some issues in the construction of IORs.

Server Side Configuration

This section describes the features of the gateway that support servers behind the firewall. In particular, it describes how incoming invocations are processed, and how services can be made visible to potential clients.

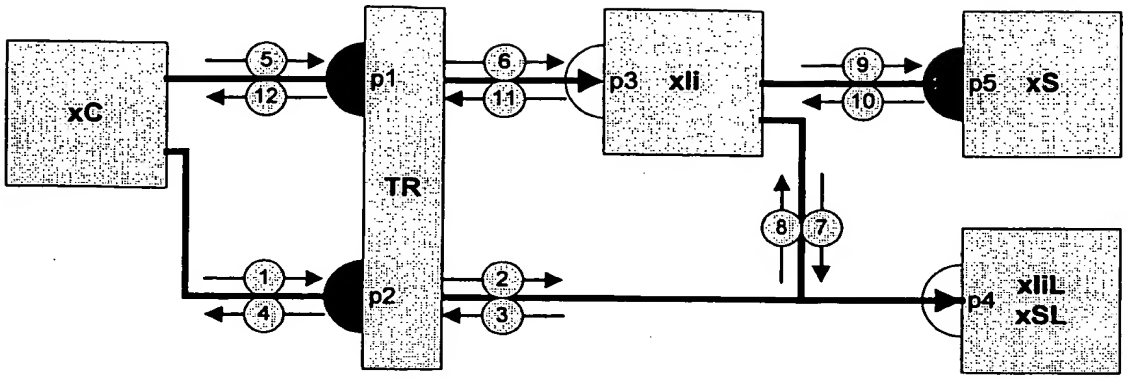
Inbound Invocations

In the description that follows, some CORBA client xC outside the firewall will invoke a server xS inside the firewall, behind the object gateway. (Behind in the sense that the server is on the side from which the gateway is controlled.)

Invocation steps and components

Figure 6 shows the components involved in an incoming invocation from a client. See "Nomenclature" on page 32 for an explanation of the conventions used in the diagrams.

Figure 6 Inbound Invocation



The components may be running on separate hosts. In the following descriptions, the host will be identified by the component name prefixed with 'h' (e.g. hxS is the host on which xS runs).

The sequence of events for an inbound invocation starts with xC holding the object reference ref=IOR[hTR:p2:kxI] and making the invocation ref→op(args). Initially, none of the connections is in place; for subsequent invocations, existing connections may be re-used.

1. xC connects to hTR:p2 and sends request kxI:op:args
2. TR connects to hxIiL:p4 and sends request kxI:op:args - TR needs to know to connect to hxIiL:p4 for connections accepted on p2
3. xIiL replies with LOCATION_FORWARD to IOR[hTR:p1:kxI] - xIiL needs to know kxI→IOR[hTR:p1:kxI], this is the standard information that the object locator knows, except that the port is p1 rather than p3. xIi must have created this IOR and registered it with xIiL.
4. TR replies with LOCATION_FORWARD to IOR[hTR:p1:kxI]
5. xC connects to hTR:p1 and sends request kxI:op:args
6. TR connects to hxIi:p3 and sends request kxI:op:args - TR needs to know to connect to hxIi:p3 for connections opened on p1, xIi will have instructed TR to do this as described under "Initialisation" below.
7. xIi connects to hxSL:p4 and sends request kxS:op:mapoi(args) - xIi knows kxI→IOR[hxSL:p4:kxS] because kxI=kio(IOR[hxSL:p4:kxS]) where kio is a function that wraps an IOR and some other information into an object key, and xIi has functions to verify kxI and extract the IOR. mapoi is a function that replaces IORs in their outside form

with IORs that will work on the inside. These functions are described in more detail under "Mapping Object References" on page 22.

8. xSL replies with LOCATION_FORWARD to IOR[hxS:p5:kxS] - xSL needs to know kxS→IOR[hxS:p5:kxS], this in the standard object locator information.
9. xIi connects to hxS:p5 and sends request kxS:op:mapoi(args)
10. xS performs service and replies status:results (if status signifies an exception, results is the marshalled exception)
11. xIi replies status:mapio(results), mapio is a function that replaces IORs in their inside form with IORs that will work on the outside
12. TR replies status:mapio(results)

The mapoi and mapio functions are described in more detail below under Mapping Object References.

Options

If the object reference held by xC is IOR[hTR:p1:kxI,hTR:p2:kxI], and xC tries the profiles in order, the process will start at step 5. If xIi is running and TR is relaying p1 to p3, the invocation will proceed as shown. If the relay can be made to re-use the same port number for a service if it is available, connections can be made slightly more quickly. If the port is not available, the relay can use any port, but in this case, the chosen port number must be passed back into the relocation mechanisms.

There may be a packet filter or other firewall component restricting the ports that can be used to connect to TR from the outside. In this case, TR must choose a port for p1 from the set of ports that the external component allows. Either TR or xIi may be configured to know which ports to use. If xIi rather than TR has the information, then a protocol will be required to deal with the case where the suggested port is already in use, and a new suggestion is required.

If TR can be guaranteed to obtain all of the ports allocated for use from outside, and the mapping of external ports to interceptors is static for the lifetime of persistent object references to those interceptors, the object locator and its relocation mechanisms are not required.

At step 3, xIiL may start xIi if it is not already running. The mechanism for doing this is described below in the section "Starting Interceptor Servers Dynamically".

At step 7, if the IOR embedded in kxI was IOR[hxS:p5:kxS,hxSL:p4:kxS] and xIi tries the profiles in order, the process will continue from step 9 rather than step 7

Properties of the configuration

The only directly accessible ports are those on which TR is listening: p1 and p2.

Indirectly accessible from the outside are the ports to which TR will connect: p3 and p4.

TR must be running on host hTR and listening on port p2 which are in the external form IORs in order to receive the object locator requests.

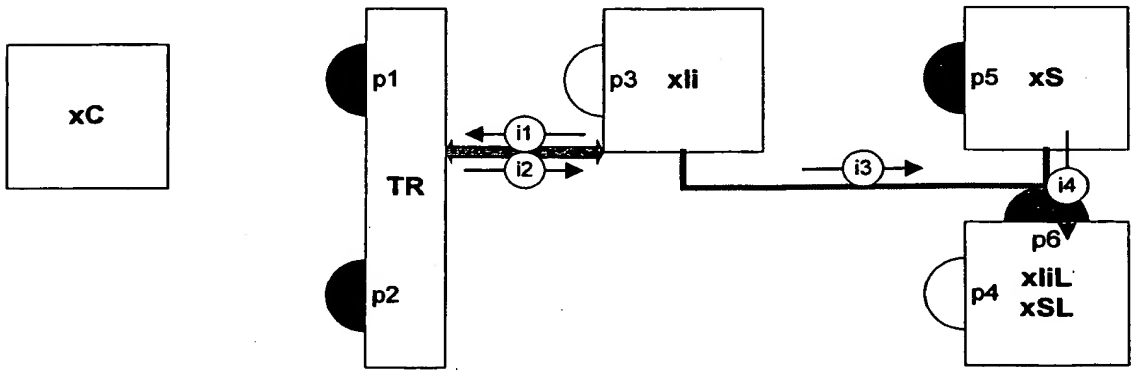
The interceptor xIi listens for requests on port p3. Incoming connections on port p1 are relayed to p3, therefore the external form of reference can contain a profile referring to p1 (rather than p3), as well as containing a profile referring to p2 (rather than p4).

For persistent outside references to work, port p2 must be fixed (e.g. the object locator port). The locator mechanisms will fix up other ports, but re-using old ports will have a slight efficiency advantage.

Initialisation

The invocation described above assumes that various mappings are in place, and IORs contain particular information. Some initialisation steps establish this position.

Figure 7 Initialisation steps



1. xIi instructs TR to establish a relay from p1 to p3
2. TR responds with actual value of p1 used
3. The IOR for xIi is registered with xIiL, but with p1 rather than p3 as the port.
4. The IOR for xS is registered with xSL, this is a normal registration.

Steps 3 and 4 will have responses, but these are not shown in the diagram since there is no significant new data in them.

Steps 1, 2 and 3 must occur in order, but step 4 is independent of the others. Step 4 could occur before the others or at any time before step 8 in Figure 6 (the locator responding with the location of xS). In particular, xS may be started in response to the request from xIi for that service (step 7 in Figure 6).

The instructions to TR from xIi, and responses to those instructions, are sent via pipes. The idea is to ensure that unauthorised programs cannot instruct TR to open connections. In order to establish the pipe connections, xIi is started by TR. It is important that TR knows that xIi has not been modified. xIi is relatively more sensitive than other components since it can instruct TR to establish incoming connection relays.

Note that xIiL is shown as listening on two ports. This is because one - p4 - is exposed via TR for invocation attempts to be remapped, and administrative operations should not be exposed through that port. All administrative requests should arrive via p6, and this is the port that should appear in the IOR for the object locator (which can be found in /etc/opt/orbplus.)

Interceptors such as xIi construct the IORs that are passed out to external clients. When creating the external reference for xS, xIi must use p2 as the locator port, and should use p1 as the hint port.

Which component needs to know what?

xIiL needs to know p4, the port on which it is to listen. Conventionally, the object locator listens on a fixed port, but if xIiL is used only via TR, p4 need not be fixed, but in this case, the port used must be made known to TR and xIiL. Static configuration is the simpler option.

xIiL does not need to know p2, provided that xIi rather than xIiL creates the external form IORs, or can substitute port numbers in the relevant places.

xIi needs to know p2 as locator port for inclusion/substitution in IORs.

xIi needs to know p1 as substitute for its own p3 both for use in the IOR passed to xIiL for relocation, and for inclusion in another profile in external form IORs if the short-circuit mechanism is required.

TR needs to know the p1-p3 mapping - accept a preferred p1 value from xIi, but it must be able to report the actual value used back to xIi.

TR needs to know the p2-p4 mapping, this may be statically configured

Potential exposures

This section describes some possible attack strategies, and identifies the feature of the configuration that defeats them. Its primary aim is to explain why certain features are included.

If xIiL is also the object locator for inside services, what happens?

xIiL knows the kxS→IOR:[hxS:p5:kxS] mapping.

A connection to hTR:p2 and request message kxS:op:args will respond with LOCATION_FORWARD to IOR[hxS:p5:kxS] through the TR connection to xIiL.

An attempt to connect to hxS:p5 from outside will fail.

A connection to hTR:p1 with request message kxS:op:args will be forwarded to hxIi:p3 and then fail. It fails because xIi rejects requests for unknown virtual servers (and the VS names are distinct).

The attacker learns that kxS is at least partially valid as an object key. If the attacker understands ORB Plus IORs, it can discover that the virtual server name in the object key is valid (it is not clear if more is revealed).

The attacker invokes an administration operation (e.g. register_sp, set_command) on the object locator through port p2.

xIiL must use separate ports for admin (p6) and for redirecting invocations (p4). Invocations sent to p2 are forwarded to p4. If xIiL keeps these functions separate, and does not establish a relay to p6, the request will be rejected. N.B. this requires a modified object locator.

Generic Interceptors and Clients using the Dynamic Invocation Interface

The CORBA Dynamic Skeleton Interface (DSI) and Dynamic Invocation Interface (DII) makes it possible to implement a generic interceptor that can handle interfaces with any arbitrary IDL. This is in contrast to type-specific interceptors that include stub and skeleton code generated by an IDL compiler, and specific implementations of the operations defined in the interface.

Clients may also use DII to invoke a service. In particular, the client of an inbound invocation may be a generic interceptor in an outbound invocation through another firewall as described below under Combined Configuration.

In order to support DII, a get_interface operation can be invoked on any interface. This operation returns a reference to an InterfaceDef - an interface supported by a standard service called the Interface Repository. The client can then invoke operations on the returned InterfaceDef to explore the type while constructing a request with the correct parameter types for the operation. Figure 8 shows what happens when a client using DII invokes a service through a Generic Interceptor that also uses DII.

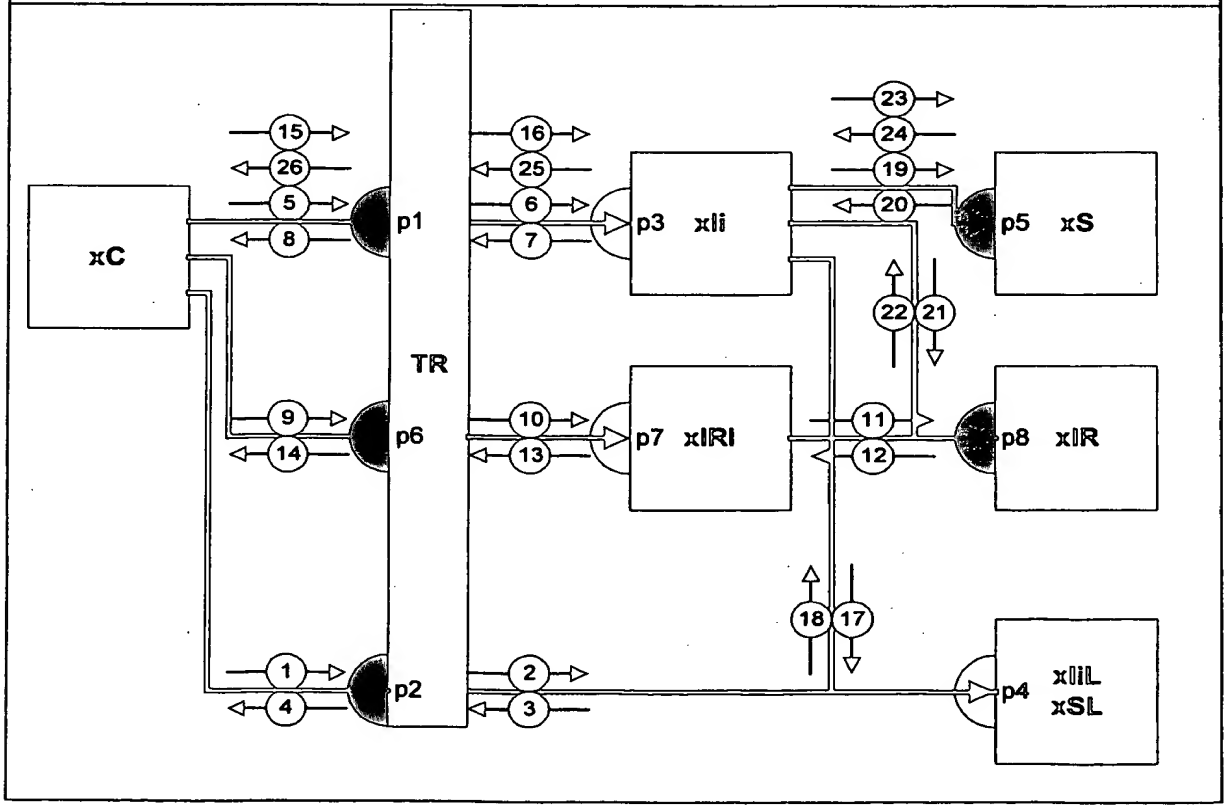
Invocation steps and components

Figure 8 shows the components involved in an incoming invocation from a client that uses DII. In addition to those in Figure 6 there are:

xIRI: the interceptor for InterfaceDef services in xIR

xIR: the Interface Repository which holds the definition of the 'x' interface

Figure 8 Interface Repository and Clients using DII



The sequence of events for an inbound invocation from a client xC using DII starts with xC holding the object reference ref=IOR[hTR:p2:kxI] and enquiring about interface so that it can construct a request. xC does this by making the invocation ref→get_interface().

1. xC connects to hTR:p2 and sends request kxI:get_interface: since it is using DII and needs information about the interface in order to construct a request.
2. TR connects to hxIiL:p4 and sends request kxI:get_interface: - TR needs to know to connect to hxIiL:p4 for connections accepted on p2 as before
3. xIiL replies with LOCATION_FORWARD to IOR[hTR:p1:kxI] - xIiL needs to know kxI→IOR[hTR:p1:kxI] as before
4. TR replies with LOCATION_FORWARD to IOR[hTR:p1:kxI] - these first four steps are equivalent to the first four steps of the inbound invocation described above. The operation is different, but this has not yet been used.
5. xC connects to hTR:p1 and sends request kxI:get_interface:
6. TR connects to hxIi:p3 and sends request kxI:get_interface: - TR needs to know to connect to hxIi:p3 for connections opened on p1 as before
7. xIi replies with NO_EXCEPTION:mapio(xDef) - xDef is a reference to an object in xIR that describes interface 'x', mapio translates this to the externally usable form of a reference to an interceptor in xIRI. Let xDef be IOR[hxIR:p8:kxD], then mapio(xDef) is IOR[hTR:p6:kio(IOR[hxIR:p8:kxD])] where kio is a function that wraps the destination IOR and some other information into an object key.
8. TR replies with NO_EXCEPTION:IOR[hTR:p6:kio(IOR[hxIR:p8:kxD])]
9. xC connects to hTR:p6 and sends request kio(IOR[hxIR:p8:kxD]):ir_op:ir_args invoking some interface repository operation
10. TR connects to hxIRI:p7 and sends request kio(IOR[hxIR:p8:kxD]):ir_op:ir_args - TR needs to know to connect to hxIRI:p7 for connections opened on p6
11. xIRI connects to hxIR:p8 and sends request kxD:ir_op:mapoi(ir_args) - note that the arguments to the interface repository operation originate on the outside, and so mapoi is used to convert any object references in the arguments to forms that will work on the inside.
12. xIR replies status:results as appropriate to the operation
13. xIRI replies status:mapio(results) - the results will be sent outside so inside to outside mapping function mapio is used.

14. TR replies status:mapio(results); steps from 9 are repeated until xC has enough information about the interface, operation, and parameters
15. xC uses connection established at step 5 to hTR:p1 and sends request kxI:op:args - apart from the connection already existing, this is step 5 from the previous inbound invocation.
16. TR uses connection established at step 6 to hxIi:p3 and sends request kxI:op:args.
17. xIi connects to hxSL:p4 and sends request kxS:get_interface: - xIi is a generic interceptor that uses DII. Like the client at step 1, it needs to obtain information about the interface in order to construct a request. A generic interceptor also needs this information in order to understand the arguments in the incoming request. xIi knows kxI→IOR[hxSL:p4:kxS] because kxI=kio(IOR[hxSL:p4:kxS]) where kio is a function that wraps an IOR and some other information into an object key, and xIi has functions to verify kxI and extract the IOR.
18. xSL replies with LOCATION_FORWARD to IOR[hxS:p5:kxS] - xSL needs to know kxS→IOR[hxS:p5:kxS]
19. xIi connects to hxS:p5 and sends request kxS:get_interface: - this is the retry of step 17 with the target defined by the IOR received in step 18.
20. xS replies with NO_EXCEPTION:xDef where xDef is IOR[hxIR:p8:kxD] as in step 7
21. xIi connects to hxIR:p8 and sends request kxD:ir_op:ir_args - unlike step 11, the arguments originate on the inside at xIi, so no mapping function is used.
22. xIR replies status:results as appropriate to the operation - steps from 21 are repeated until xIi has enough information about the interface, operation, and parameters
23. xIi uses connection established at step 19 to hxS:p5 and sends request kxS:op:mapoi(args) - these are the operation and arguments sent from the client at step 15. Since the arguments originate on the outside, the outside to inside mapping function mapoi is used.
24. xS performs service and replies status:results
25. xIi replies status:mapio(results) - the results will be sent outside so inside to outside mapping function mapio is used.
26. TR replies status:mapio(results)

Options and Issues

xIRI does not support all of the interface repository interfaces. It supports only the InterfaceDef interface, and restricts callers to the describe_interface operation.

As for the previous incoming invocation example, the round trips to the object locator can be avoided if IORs have appropriate additional profiles.

At step 7, xIi responds to the get_interface operation. A normal server would find its own Interface Repository, and invoke it to look up the appropriate InterfaceDef. The server xS does this at steps 19-20. If xIi uses its own interface repository then xIi and xS must be using the same interface repository, or at least interface repositories that contain the same definitions. This standard response to get_interface is provided by the ORB, and no application intervention is required.

Propagating invocations across multiple firewalls will require that all the interceptors be able to respond correctly to get_interface. It is not practical to require that interface repositories be synchronised across multiple independent organisations. Each interceptor must intercept the get_interface operation in the same way as other operations and invoke get_interface on its target interface. Ultimately, the server will use the standard process to obtain an interface to its own interface repository.

The interceptors, such as xIi, will also need the information provided by the get_interface operation (steps 17-22 above). If xIi captures the results of the invocations from xC, xIi need not invoke the interface repository later - i.e. steps 17-22 above will not be needed. xIi will also be able to respond immediately using the saved information to a subsequent get_interface from xC or any other client.

Externally Visible Naming Service (see also Annex A)

The simple strategy for making services visible is to create a context that is to contain the externally visible services. If an external client can invoke the operations of that context, the IOR mapping will make the services in that context usable externally.

The outside Initial Naming Context IOR is the mapping of the IOR for the context that contains externally visible services. The requirement is for an interface to the inside to outside mapping function of the generic interceptor that can be invoked with the inside reference to generate the outside reference. The external version of the interface can then be stringified, and passed to an external client.

There are two problems with this simple unmanaged strategy:

1. The updating operations of the naming context are available to the external client

2. There is no way to revoke services once they have been passed to outsiders.

These problems can be overcome by using a type-specific interceptor for Naming. This interceptor does not need to maintain a state store for the services that have been made visible, it can still use a standard naming context, and just filter the invocations. The updating operations can simply return a NO_PERMISSION exception, or can apply access control to permit some updating of certain contexts. This may be a way to provide certain kinds of remote administration. A generic interceptor with access control at the operation level may also be able to give the required degree of control.

Control over revocation can be added by including a tag in the external form of the object references that effectively links them back to some initial reference. A specialised interceptor for the initial naming context is a suitable place to insert new tags. Interceptors would be instantiated with a tag, and propagate that tag into the external form references that they generate. Current validity of the tag can be checked at interceptor instantiation time; immediate revocation can be achieved by iterating through all interceptors, instructing those with the matching tag to shut down. This is discussed in more detail under "Mapping Object References" on page 22.

Starting Interceptor Servers Dynamically

The ORB Plus object locator can start server processes dynamically. It does this by invoking a command that has been registered for a server process. This can be used to avoid having rarely used servers running all the time - server processes can time-out and shut down, then restart on demand. This can also avoid the need to include servers in system restart configurations. Servers registered with the object locator can be left to start when they are first called.

This technique can be applied to interceptor processes. Starting a new type of interceptor server while the gateway is running presents the same issues as starting interceptors from the object locator.

There are two possible strategies for starting interceptors dynamically:

1. Have the object locator start the interceptor and provide a trustworthy way for a valid interceptor to pass instructions to the trusted relay.

2. Have the object locator start a program that instructs the trusted relay to launch an interceptor.

Both of these strategies require that a program be able to establish communication with the trusted relay.

In the first strategy, the trusted relay has no way to verify the instructions. It must have sufficient confidence that any program able to establish communication is also authorised to issue instructions to establish a connection relay. The mechanism that allows the program to establish communication should be disabled if the executable file is modified. If it is possible to arrange that interceptors have a privilege that allows them to connect to the trusted relay's command input, this would achieve the objective.

In the second strategy, the trusted relay can consult a configuration file, or some other resource, to obtain the parameters it needs to ensure that the interceptor is acceptable, and unmodified. Once the check has been passed, the trusted relay can start the interceptor with a pipe connection for transmitting connection relay requests.

Further study is needed to determine which of these strategies to adopt. The choice may depend on available features of the underlying system, and it may be appropriate to use different strategies on different operating systems.

Client Side Configuration

This section describes the features of the object gateway that support clients behind the firewall. In particular, it describes how outgoing invocations are processed, and how services visible outside the firewall can be made visible inside.

Outbound Invocations

In the description that follows, some CORBA server yS outside the firewall will be invoked by a client yC that is inside the firewall, behind the object gateway. (Behind in the sense that the client is on the side from which the gateway is controlled.)

Invocation steps and components

The diagram below shows the components involved when client yC invokes server yS.

yC: the client for service 'y'

yIo: the interceptor for outbound invocations of service 'y'

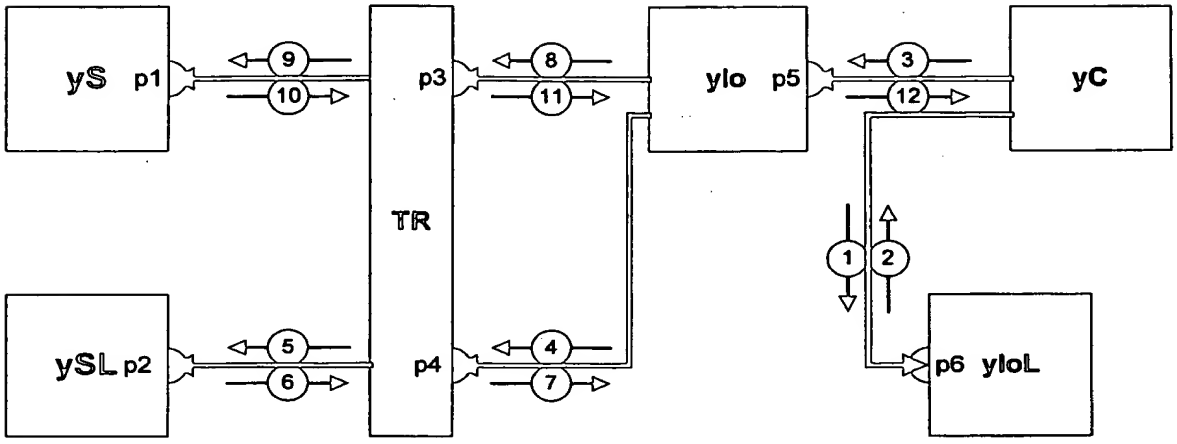
yIoL: the object locator for 'yIo'

TR: the Trusted Relay

yS: the server for service 'y'

ySL: the object locator for yS

Figure 9 Outbound Invocation



The components may be running on separate hosts. In the following descriptions, the host will be identified by the component name prefixed with 'h' (e.g. hyC is the host on which yC runs).

The sequence of events for an outbound invocation starts with yC holding the object reference ref=IOR[hyIL:p6:kyI] and making the invocation ref→op(args). Initially, none of the connections is in place; for subsequent invocations, existing connections may be re-used.

1. yC connects to hyIL:p6 and sends request kyI:op:args
2. yIoL replies with LOCATION_FORWARD to IOR[hyIo:p5:kyI] - yIoL needs to know kyI→IOR[hyIo:p5:kyI], this is standard CORBA relocation
3. yC connects to hyIo:p5 and sends request kyI:op:args
4. yIo connects to hTR:p4 and sends request kyS:op:mapio(args) - yIo knows kyI→IOR[hySL:p2:kyS] because kyI=koi(IOR[hySL:p2:kyS]) where koi is a function that wraps an IOR and some other information into an object key, and yIo has functions to verify kyI and extract the IOR. (Note that koi wraps an outside IOR into a key for use inside, whereas kio which was introduced earlier wraps an inside IOR into an object key

for use outside. These functions are described in more detail under "Mapping Object References" on page 22. yIo also needs to know hySL:p2→hTR:p4 (perhaps creating the mapping)

5. TR connects to hySL:p2 and sends request kyS:op:mapio(args) - TR needs to know to connect to hySL:p2 for connections accepted on p4, see "Outbound Relay Connections" below.
6. ySL replies with LOCATION_FORWARD to IOR[hyS:p1:kyS] - ySL needs to know kyS→IOR[hyS:p1:kyS], this is standard CORBA relocation
7. TR replies with LOCATION_FORWARD to IOR[hyS:p1:kyS]
8. yI connects to hTR:p3 and sends request kyS:op:mapio(args) - yI needs to know hyS:p1→hTR:p3 (perhaps creating the mapping)
9. TR connects to hyS:p1 and sends request kyS:op:mapio(args)
10. yS performs service and replies status:results
11. TR replies status:results
12. yI replies status:mapoi(results)

Options and issues

If the object reference held by yC is IOR[hyI:p5:kyI,hyIL:p6:kyI], and the profiles are processed in order, the invocation will start at step 3.

At step 4, if yI knows kyI→IOR[hyS:p1:kyS,hySL:p2:kyS], and processes the profiles in order, it will proceed from step 8 instead of step 4.

At steps 4 and 5 and again at steps 8 and 9, the interceptor needs to connect to the relay rather than the host and port specified in the IOR, and the relay needs to make the onward connections. The interceptor is passing instructions to the relay, and the relay must have a way to know that the instructions are trustworthy.

Outbound Relay Connections

There are two possible approaches to establishing outbound connections through the relay.

1. A socks-like approach where a connection is made to some nominated port, and the destination host and port are sent as an in-band prefix to the main communication.
2. An out of band control channel is used to set up the outbound connection relay.

In either case, the trusted relay needs to know that the instructions are from a source authorised to specify outbound connection relays. Outbound connections may be considered less sensitive than inbound connections, but one of the features of VirtualVault is that "inside" applications cannot connect to the outside network. A relay that accepts instructions from any inside process removes this feature.

The trusted relay can obtain the required assurance either by inspecting the instructions, or by knowing that they were delivered over a trustworthy connection. The trusted relay could validate the instructions by checking a digital signature, but this would introduce signature checking and the related certification issues into TR. To avoid introducing this complexity into TR, the discussion will focus on using a connection that is trustworthy in the sense that it cannot be intercepted, and is to a known to originate at an authorised component.

If the destination instructions are specified over a trustworthy connection, the socks-like approach requires that all outbound traffic go via the trustworthy connection. It is not clear at present whether this is a strength or a weakness.

With the out-of-band approach, the trusted relay would be instructed to establish a relay to some specified host and port, and would reply with the port number on which it is listening for the connection from the interceptor. If the interceptor for outbound invocations is combined with the interceptor for inbound invocations then the same relay control channel can be used for establishing inbound and outbound connection relays. One disadvantage of using an out of band control system is that the connection relay exists independently of a particular connection, and a garbage collection strategy will be required to prevent a build-up of dormant relays.

Making External Services Visible to Clients

For a client to invoke an external service, it must obtain a reference to an interceptor for the external service. The IOR mapping functions will construct additional references once a first reference has been obtained. Given an internal form reference to an external naming context, resolving names in that context will provide internal form references to the services. The requirement is for an interface to the outside to inside mapping function of the generic interceptor, and an external form reference to the required external context. An internal

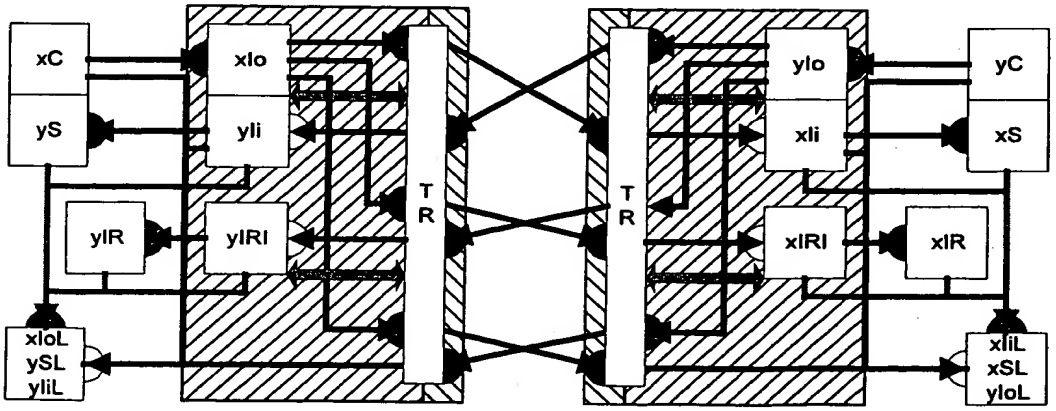
reference to the external context can then be created and bound to some suitable name in an internal naming context for use by clients.

The reference to the external context, and references derived from the use of this service, may need to be revoked. This may include internal services that have been made available externally as parameters to outbound calls. If clients are given the use of an interceptor to the root of the external hierarchy then the revocation will be all-or nothing. By creating an internal 'shadow' of the external naming structure, a finer grained revocation mechanism can be used. This will be described below under "Tags and Revocation" on page 27.

Combined Configuration

In the general case, there will be both clients and servers behind each firewall, and there will be firewalls at both ends in each interaction. This leads to the combined configuration shown in Figure 10. The components are designed to be deployed on a trusted operating system, with Mandatory Access Control being used to prevent unprivileged processes from having access to both networks. Figure 10 also shows the components that must run on the trusted hosts on a shaded background. The configuration shown here uses a simple inside/outside split as is used in VirtualVault.

Figure 10 Combined Configuration



TR the trusted relay is the only component that requires privileges. As was discussed above, it may be appropriate to grant some privilege to interceptors. None of the other components need to run on the trusted OS, nor does any need any special ORB features. The end clients

and servers, the Interface Repository and the Implementation Repository (object locator) are standard components or application components built on standard ORBs.

Mapping Object References

The descriptions above have used the functions mapio, mapoi, kio and koi to show where object reference mapping occurs. This section describes the mapping in more detail.

Some information has already been given in the description of invocations from DII clients.

mapio and mapoi are used on arguments and results of invocations (including exceptions). They both work by traversing the structure, replacing object references with the appropriate mapped form. The important part of each function is the mapping it applies to an object reference.

Inside to outside: mapio and kio functions

For an internal reference IOR[hi:pi:ki],

mapio(IOR[hi:pi:ki]) = IOR[hTR:pTR:kio(IOR[hi:pi:ki])]

where a relay has been, or will be, established from hTR:pTR to hIiL:pIiL, the host and port for the object locator for an interceptor that will handle incoming invocations of this interface.

kio(IOR[hi:pi:ki]) = <tag,"i",IOR[hi:pi:ki],h(<tag,"i",IOR[hi:pi:ki],S>)>

The result of kio consists of a tag which allows access to be revoked, a constant "i" that signifies that this the embedded IOR is native to the inside of the gateway that generated this reference, the IOR , and a secure hash of these elements and a secret S.

By default, interceptors are instantiated containing the tag that they will embed in the keys generated by their kio function. Specialised interceptors have the option of inventing new tags to embed in references that they pass out. The naming interceptor for the bootstrap context will use this to assign different tags to the top level services.

The secret S is used to verify that the key has not been forged. The interceptor that generates the object key, and any interceptor that unwraps the key must share the secret. All interceptors in the gateway will need to know the secret, but it need not, and should not, be known outside. There could be a different secret for each tag; the advantages and disadvantages of this approach are not known.

The secret must be stored securely, since knowing this secret makes it possible to construct an IOR that will be accpted by the gateway. When the gateway is deployed on a system with Mandatory Access Control and Multi-Level Security, the key can be stored with a sensitivity label designated for key storage, and for which gateway processes and other applications are not cleared. The trusted relay can be cleared to access the keys, and it can also provide a signing function for the interceptors that are connected to it. The existing trustworthy channel between interceptor and relay can be used to invoke the signing and signature checking function.

For a mapped external reference, mapio can remove the wrapping and return the embedded IOR if it is native to the target environment. (It might not be if there is more than one gateway and they lead to different 'outside's.) An external reference that came in through this gateway will have been mapped by the mapoi function described below. It will have the form:

IOR[hIoL:pIoL:koi(IOR[ho:po:ko])]

where

koi(IOR[ho:po:ko]) = <tag,"o",IOR[ho:po:ko],h(<tag,"o",IOR[ho:po:ko],S>)>

If the object key has the expected structure, and the hash is valid, the presence of the "o" indicates that

IOR[ho:po:ko]

is the corresponding external interface and can be returned by mapio.

If the key verifies correctly but contains "i" rather then "o", it means that the IOR is theexternal reference to an internal service and has been passed in by some out-of-band mechanism. Such a reference can be used unchanged.

Outside to inside: mapoi and koi functions

For an external reference IOR[ho:po:ko],

mapoi(IOR[ho:po:ko]) = IOR[hIoL:pIoL:koi(IOR[ho:po:ko])]

where hIoL:pIoL is the host and port for the locator for the interceptor that will deal with outbound invocations.

koi(IOR[ho:po:ko]) = <tag,"o",IOR[ho:po:ko],h(<tag,"o",IOR[ho:po:ko],S>)>

The result of koi consists of a tag which allows access to be revoked, a constant "o" that signifies that this the embedded IOR is native to the outside of the gateway that generated this reference, the IOR , and a secure hash of these elements and a secret S.

Tagging references that come in is necessary in order to propagate the tags to outgoing references in nested invocations. The tags will also make it possible to revoke references to external services that have been brought inside as well as references to internal services that have been made visible outside.

In this case, the secret S protects against forgery on the inside. Forging an inside form of an external reference would make it possible to establish a connection relay to an arbitrary outside host and port. A malicious client could forge an IOR embedding a bogus external IOR with a chosen target host and port, for example, by modifying an existing IOR. If the client uses the forged reference as an invocation target, an interceptor will be created for the bogus external service. The interceptor would instruct the relay to establish a connection to the host and port chosen by the forger. If the out-of-band control is being used, the forger can probe for the corresponding port on the relay and so establish a connection to its chosen destination. (This is an argument in favour of the socks-like in-band control strategy where all connections to the relay from the inside must come from components known to be trustworthy - i.e. interceptors.)

For a mapped internal reference, mapoi can remove the wrapping and return the embedded IOR. An internal reference that went out through this gateway will have been mapped by the mapio function described above. It will have the form:

IOR[hTR:pTR:kio(IOR[hi:pi:ki])]

where

kio(IOR[hi:pi:ki]) = <tag,"i",IOR[hi:pi:ki],h(<tag,"i",IOR[hi:pi:ki],S>)>

If the object key has the expected structure, and the hash is valid, the presence of the "i" indicates that

IOR[hi:pi:ki]

is the corresponding internal interface and can be returned by mapoi.

If the key verifies correctly but contains "o" rather then "i", it means that the IOR is the internal reference to an external service and has been passed out by some out-of-band mechanism. Such a reference can be used unchanged.

Inbound invocation target

An inbound invocation will arrive at an interceptor presenting the object key constructed by a previous call of kio and embedded into an IOR by mapio. It will have the form

kio(IOR[hi:pi:ki]) = <tag,"i",IOR[hi:pi:ki],h(<tag,"i",IOR[hi:pi:ki],S>)>

if the required interceptor instance is not currently in place, this object key will be passed to the activator which can instantiate the interceptor.

The activator is part of an interceptor server and has access to the secret 'S'. It can verify the key and check that the key contains the "i" that indicates that the service is native to the inside of this gateway. The activator can then extract the tag and the IOR. If the tag is currently valid (i.e. has not been revoked), the activator can instantiate an interceptor for the IOR and tag.

Outbound invocation target

An outbound invocation will arrive at an interceptor presenting the object key constructed by a previous call of koi and embedded into an IOR by mapoi. It will have the form

koi(IOR[ho:po:ko]) = <tag,"o",IOR[ho:po:ko],h(<tag,"o",IOR[ho:po:ko],S>)>

if the required interceptor instance is not currently in place, this object key will be passed to the activator which can instantiate the interceptor.

The activator is part of an interceptor server and has access to the secret 'S'. It can verify the key, and check that the key contains the "o" that indicates that the service is native to the outside of this gateway. The activator can then extract the tag and the IOR. If the tag is currently valid (i.e. has not been revoked), the activator can instantiate an interceptor for the IOR and tag.

Multiple profiles

The mapping functions have been shown manipulating IORs with only a single profile. Additional profiles can be added both to avoid the use of the object locator, and to make the

IORs usable if they are passed by some means other than the gateway (e.g. stringified on a web page or in e-mail).

If mapio includes the original profile in the constructed IOR, it will work if it is passed back inside by some alternative route:

mapio(IOR[hi:pi:ki]) = IOR[hi:pi:ki, hTR:pTR:kio(IOR[hi:pi:ki])]

if the original IOR had several profiles they can all be copied. The reverse transformation that mapoi applies will need to remove duplicated profiles in this case.

To avoid the object locator, an additional profile can be added:

mapio(IOR[hi:pi:ki]) = IOR[hTR:pTRI:kio(IOR[hi:pi:ki]), hTR:pTR:kio(IOR[hi:pi:ki])]

where pTRI is the port on which TR is listening for connections that go the interceptor for the service.

Tags and Revocation

Tags are embedded into the mapped IORs so that the gateway can determine the origin of the reference that is being used to make an invocation. Some references are given distinct tags as starting points; any references passed in invocations will be given the same tag, and so can be traced back to one of those starting points. The tags can be used to revoke access to all references passed as a result of using some initial reference, without disturbing other references.

Tagging by externally visible name

The simple strategy for making services visible was described under "Externally Visible Naming Service" on page 16. The services are bound in a context for which an interceptor is created, and the reference to the interceptor is published. In general, there can be a hierarchy of contexts under that initial context. The references bound in that hierarchy can be tagged with their names relative to the root of that hierarchy by a specialised naming interceptor.

The specialised naming interceptor performs special mappings on the object references that are passed as parameters. In particular, the reference returned by the 'resolve' operation is the mapped form of the internal reference, with a new tag chosen by the naming interceptor. The interceptor for the root of the externally visible hierarchy uses the name passed as a parameter

to 'resolve'. It also ensures that references to naming contexts are mapped to external references to other instances of the specialised naming interceptor.

When an operation is invoked on a sub-context for the first time, a new specialised naming interceptor will be activated with the tag that was the name of that context relative to the root of the externally visible hierarchy. When 'resolve' is called on this new interceptor, it will construct a new tag by concatenating its activation tag with the name passed as a parameter. The external root interceptor needs no special code, it just needs to be activated with some chosen 'root' tag, and this will be propagated as a prefix to all the tags of the sub-contexts.

With this tagging system, services, and all their derived references, are revoked by unbinding them from the externally visible context.

Tagging imported services

Where external services are made visible to clients, the references also have tags that can be used to revoke access. References to some initial set of interceptors for external services are bound in some internal context. Using the names in this internal context as the tags for the interceptors leads to the simple revocation process described below.

In this case, interceptors used for naming contexts should not have special tagging behaviour. The names being resolved are external names, and do not correspond to internal bindings that can be controlled.

Revocation

Since interceptors are activated on demand by invocations, the revocation mechanism must block new interceptor activations. Revocation must apply to both inbound and outbound interceptors, and the process is similar in both cases.

References derived from exported services

With the tagging scheme described above, this is simple for references derived from a service that was made externally visible. The activator is presented with a tag that is the name of the originating service relative to the externally visible service context, perhaps with some 'root' tag prefix. The activator removes the prefix, then attempts to resolve the name. If the resolve succeeds, the service is still available and activation can proceed.

The process can be simplified still further by making the 'root' tag prefix the name of the externally visible service context relative to the internal initial naming context. The activator then just resolves the name, and does not need special knowledge of the external service context.

Immediate revocation of active interceptors can be achieved by mapping through all the interceptors, shutting down all those that were activated with the tag that has now been revoked.

References derived from imported services

References derived from an external service are tagged with the name by which the interceptor for the external service is bound in the internal naming service. Validity of the tag can be checked by resolving the name. The validation process is the same as for references derived from exported services.

Reincarnation of references

With the tagging scheme described above, a name cannot be re-used for a new service if there was an old service with that name, which was revoked. If the name were to be re-used, references to the old service, and other references passed by using that reference, would become usable again.

This reincarnation of revoked references can be prevented by including in the tag both the name and an object key. The object key is taken from either the exported service, or the interceptor for the imported service. Since the interceptor activator already resolves the name in the tag, it will have the object key available for the additional check if the resolve succeeds. If the name is re-used for a new service with a new object key, the tags will not match and the revoked references will remain revoked.

If the original service is reinstated, or a new service starts with the same object key, then the old references will once again become valid. This is consistent with the persistence of object references.

Performance of application specific and generic interceptors

If generic interceptors use the Dynamic Skeleton Interface and Dynamic Invocation Interface in the conventional way, the processing requirements for each invocation will be large. This will make it difficult to achieve a high performance gateway.

Application specific interceptors process an IDL type known in advance, and are able to use the more efficient compiled skeletons and stubs.

In order to avoid implementing application specific interceptors for every type used in an application, the performance of the generic interceptors must be brought up to the level of the application specific interceptors.

Dynamic Stub and Skeleton Generation

The dynamic interfaces use a definition of the type of an incoming request that contains the same information as the IDL for the type. The IDL is normally processed by an IDL compiler into source code that is then compiled into object code. The definition available to the dynamic interfaces can equally well be processed by an IDL compiler. This can be done either by generating the textual form for the standard IDL compiler, or by adapting the IDL compiler to work directly from the available definition. The latter process will usually be more efficient.

The IDL compiler may generate conventional source code that is then compiled and linked into the gateway. Alternatively, the IDL compiler may generate an intermediate code that can be interpreted efficiently by the gateway to perform the required functions. Both dynamic loading of libraries, and the use of intermediate codes (usually called byte-codes) are well known techniques.

Since it is already known that the interceptor functions will perform object reference mapping and then call the target service, the full functions normally provided by stubs and skeletons are not required. There is no need to present the arguments in the form expected by the language mapping. The generated code can operate directly upon the structures presented to the skeletons to produce the output normally generated by stubs. Operating at this level will be ORB-specific, but the IDL compiler for the ORB is specifically designed to operate at this level, and can be adapted to the task.

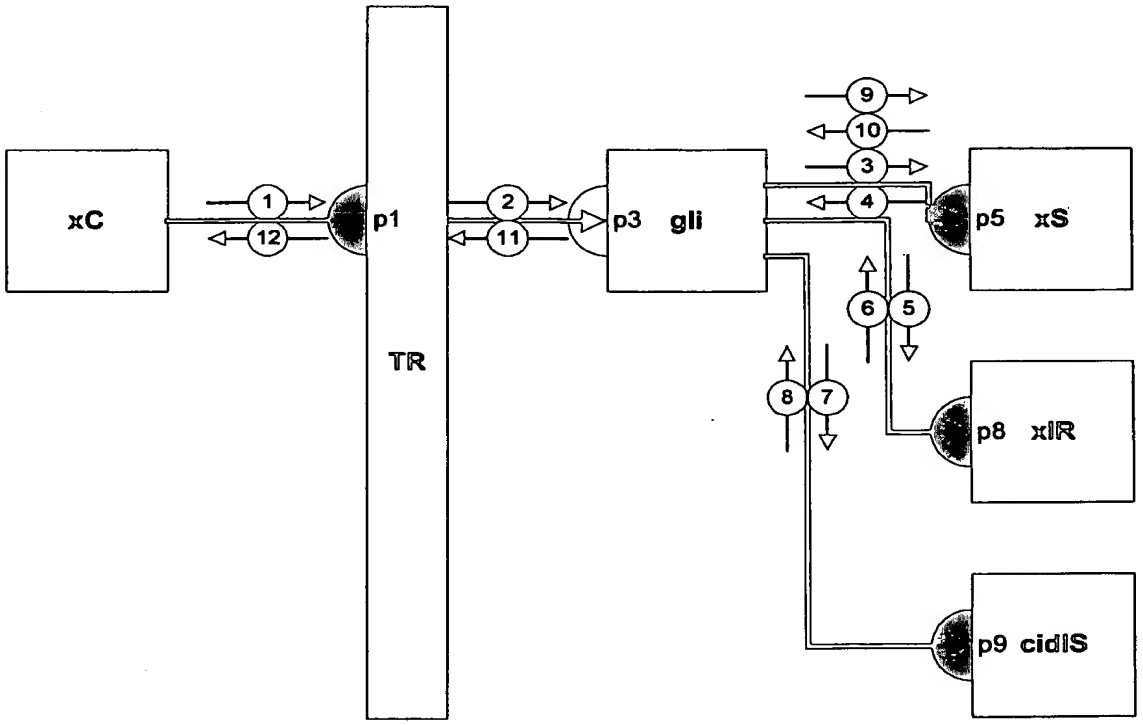
Figure 11 below illustrates the dynamic stub and skeleton generation process. This is a variant of the process illustrated in Figure 8 *Interface Repository and Clients using DII*, in which the interactions with the object locator, and DII client interactions with the interface repository have been omitted for clarity. The new components are:

gIi: a generic interceptor for inbound invocations

cidlS:the IDL compiler server

The IDL compiler is shown as a CORBA service in order to illustrate when it is used. In practice, it may be incorporated into the generic interceptor, or invoked as a program, depending on the available language and operating system facilities.

Figure 11 Dynamic Stub and Skeleton generation and loading



The sequence of events for an inbound invocation from a client xC starts with gIi holding a reference IOR[hcidlS:p9:kcidl] to the IDL compiler server, and xC holding the object reference ref=IOR[hTR:p1:kxI], and making the invocation ref→op(args).

1. xC connects to hTR:p2 and sends request kxI:op:args.
2. TR connects to hgIi:p3 and sends request kxI:op:args.

3. gIi does not have an implementation registered for kxI on the first call, and so passes the key to its activator function. The activator in gIi connects to hxS:p5 and sends request kxS:get_interface: - gIi needs to obtain information about the interface in order to generate the stubs and skeletons that will be used to process the request. gIi knows kxI→IOR[hxS:p5:kxS] because kxI=kio(IOR[hxS:p5:kxS]) where kio is a function that wraps an IOR and some other information into an object key, and gIi has functions to verify kxI and extract the IOR.
4. xS replies with NO_EXCEPTION:xDef - xDef is IOR[hxIR:p8:kxD], a reference to an 'InterfaceDef' object in xIR that describes interface 'x'.
5. gIi connects to hxIR:p8 and sends request kxD:describe_interface:
6. xIR replies NO_EXCEPTION:xDesc where xDesc is the 'FullInterfaceDescription' for 'x'.
7. gIi connects to hcidlS:p9 and sends the request kcidl:compile:xDesc, requesting compiled skeletons and stubs corresponding to the description xDesc
8. cidlS replies NO_EXCEPTION:xIicode, where xIicode is the compiled code for an interceptor specialised for 'x'.
9. The activator in gIi loads the code 'xIicode' and registers this as the implementation for the key kxI. The invocation received in step 2 is passed to this new interceptor for processing. The new interceptor in gIi uses the connection established at step 3 to hxS:p5 and sends request kxS:op:mapoi(args) - these are the operation and arguments sent from the client at step 1. Since the arguments originate on the outside, the outside to inside mapping function mapoi is used.
10. xS performs service and replies status:results
11. gIi replies status:mapio(results) - the results will be sent outside so inside to outside mapping function mapio is used.
12. TR replies status:mapio(results)

Options and Issues

The dynamically generated interceptor can remain in place to handle further invocations, and additional instances of the interceptor can be created to handle other objects of the same type, without needing to regenerate or reload the code.

Unless there is an exceptionally large number of types in use, the generic interceptor will be able to keep the code available thus avoiding the cost of going back to the IDL compiler.

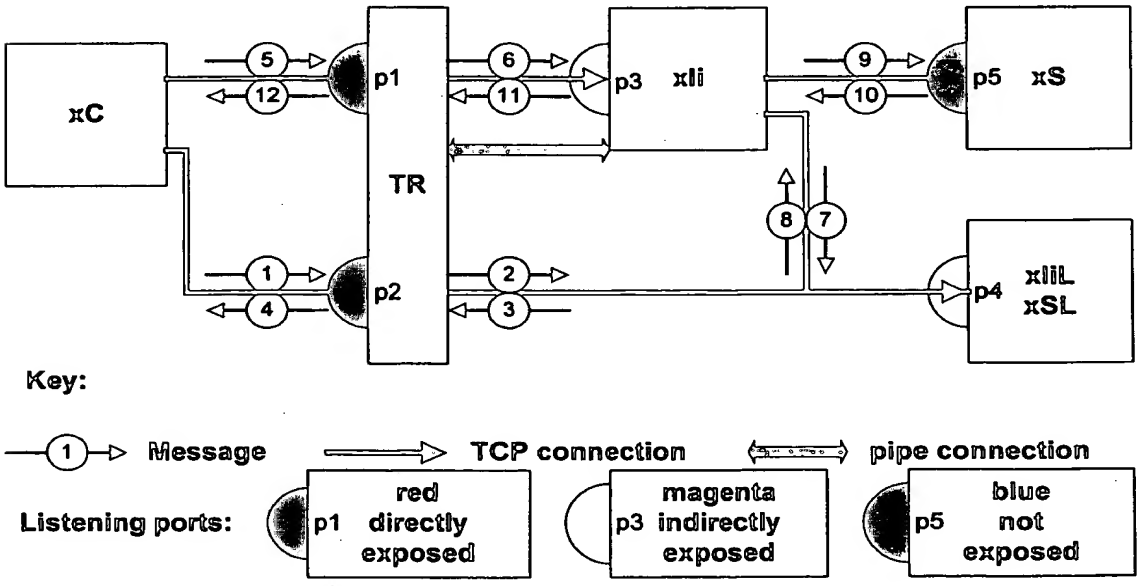
The delay on the first call can also be avoided by generating the code in advance. The server object reference is available when it is being mapped for passing as an argument or result. The code generation could be performed at any time after this up to the point where the interceptor is needed as described in the example above.

This dynamic skeleton and stub generation process can also be applied to a DCOM gateway since DCOM also defines the interfaces to objects in an Interface Definition Language that is processed by an IDL compiler.

Nomenclature

This section explains the notation used in the diagrams and associated descriptions throughout the document. Figure 12 shows an example diagram, with a key to the elements used.

Figure 12 Key to the diagram notation



The components are labelled so that the discussions that follow can refer to them. Except for TR, the trusted relay, the labels follow a convention and are of the form sF[L]. s is a letter (x or y) indicating the type of service the component handles. F specifies the function of the component: C for client, S for server, Ii for interceptor for inbound invocations, Io for interceptor for outbound invocations, IR for interface repository and IRI for interface repository interceptor . The L suffix indicates the object locator for the component. Some

components are given multiple labels, this indicates that the component performs several functions, and that these functions could be separated. The components in the diagram above are labelled following this convention:

TR: the Trusted Relay

xC: the client for service x

xIi: the interceptor for inbound invocations of service x

xS: the server for service x

xIiL: the object locator for the interceptor for inbound invocations of service x (xIi)

xSL: the object locator for the server for service x (xS) (same component as xIiL)

TCP connection arrows indicate the direction in which the connection request is sent. The arrow head is at some port that is listening, the tail is where the connection request occurs.

Ports are colour coded and shaded to indicate how exposed they are to the outside (i.e. the less trusted network, the open Internet unless the gateway is deployed at an internal organisational boundary.) Red ports are directly exposed - these occur on the 'outside' face of TR (the trusted relay), and on external components to which the gateway connects. Magenta ports are indirectly exposed - these are the ports to which the relay will connect in response to a connection from outside. Any connection with its tail on the 'inside' face of the trusted relay must have its head at a magenta port. Blue ports are not exposed to the outside, connections to them can be made only from 'inside' components. Note that the red/blue coding corresponds to the colour coding of 'outside' and 'inside' used on VirtualVault. On a host supporting sensitivity labels, the red and blue ports would have incomparable labels. The magenta ports would have the same labels as the blue ports, but be effectively accessible from the outside, so would need particular attention in the security analysis.

The discussions are organised as numbered lists, where the numbers correspond to the messages in the diagram.

References

[OMG 95] Common Object Request Broker: Architecture and Specification (revision 2.0), OMG, 1995.

[OMG 97] Common Object Request Broker: Architecture and Specification (revision 2.1), OMG, 1997.

CLAIMS

(30970139)

1. A gateway for being situated between a first network and a second network, comprising:

first interface means for receiving from the first network a message (m) intended for an object in the second network;

interceptor means for detecting any object reference(s) in the message (m) and mapping the intercepted object reference(s) to a modified form; and

second interface means for forwarding to the second network a message (m') including any modified object reference(s).

2. A gateway according to claim 1, wherein the interceptor means is configured for mapping an object reference to a modified object reference which includes both the intercepted object reference and an object reference to respective interface means associated with the gateway.

3. A gateway according to either one of the preceding claims, wherein the interceptor means is configured for incorporating into a modified object reference an identifier to indicate from which network the object reference originated.

4. A gateway according to any one of the preceding claims, wherein the interceptor means is configured for incorporating into a modified object reference a name tag associated with the identity of the interceptor means.

5. A gateway according to any one of the preceding claims, wherein the interceptor means is configured for incorporating into a modified object reference check data which is representative of at least a part of the modified object reference.

6. A gateway according to claim 5, wherein the interceptor means is configured for enacting a hash operation on said at least part of the modified object reference and a secret, the check data comprising the result of the hash operation.

7. A gateway according to claim 6, wherein the secret is stored by and only accessible by the gateway.

8. A gateway according to any one of claims 5 to 7, wherein the interceptor means is configured for:

verifying from the check data whether the object reference has crossed the gateway before; and

if the object reference has crossed the gateway before, treating the object reference as an already-modified object reference, extracting the original object reference therefrom to be forwarded to the object in the second network; or

if the object reference has not crossed the gateway before, modifying the object reference to form a modified object reference to be forwarded to the object in the second network.

9. A gateway according to any one of the preceding claims, wherein the interceptor means is configured for detecting object reference(s) having the form of an Interoperable Object Reference, IOR[host: port: key], where the host and port values represent the means for invoking the referenced object and the key comprises data to be passed to the referenced object.

10. A gateway according to claim 9, wherein the interceptor means is configured for mapping an object reference into a modified object reference having the form IOR[host x: port x: key x], wherein key x includes the object reference IOR[host i: port i: key i] and any other data incorporated by the interceptor means.

11. A gateway according to claim 10, wherein the interceptor means is configured for mapping an object reference such that the key x of the modified object reference includes:

an object reference;

an identifier to indicate from which network the object reference originated;

a name tag associated with the identity of the interceptor means; and

check data which is representative of the object reference, the identifier and the name tag.

12. A gateway for being situated between a first network and a second network, comprising:

first interface means for receiving from the second network a message (n);

interceptor means for detecting any object reference(s) in the message (n); and

second interface means for forwarding to referenced object(s) in the first network data extracted from the respective object reference(s) and intended for the referenced object(s).

13. A gateway according to claim 12, wherein the interceptor means is configured for detecting an identifier in the message (n) that indicates from which network an object reference originated.

14. A gateway according to claim 12 or claim 13, wherein the interceptor means is configured for detecting a name tag in the message (n).

15. A gateway according to claim 14, wherein the interceptor means comprises means to determine on the basis of the name tag whether the referenced object is valid and is still available for invocation.

16. A gateway according to claim 15, wherein the means to determine comprises a call to a Naming Server, the Naming Server being configurable by an authorised party by adding or removing name tags, and the presence or absence of a name tag being indicative of whether the object associated with the name tag is available or not respectively.

17. A gateway according to any one of claims 12 to 16, wherein the interceptor means is configured for verifying detected object reference(s).

18. A gateway according to claim 17, wherein the message (n) includes check data for use by the interceptor means to verify the object reference(s).

19. A gateway according to claim 18, wherein the check data is the result of a hash operation enacted on at least part of the message (n) and a secret (s1), and the interceptor means is configured for enacting a similar hash operation on the same part of the message (n) and a secret (s2) and comparing the resulting check data with the received check data to verify the object reference.

20. A gateway according to claim 19, wherein the secret (s2) is stored by and only accessible by the gateway.

21. A gateway according to any one of claims 12 to 20, wherein the message (n) includes an object key (key x) of a modified object reference, and the object key (key x) includes an object reference(s) having the general form IOR[host i: port i: key i].

22. A gateway according to claim 21, wherein the object key (key x) includes:

an object reference;

an identifier to indicate from which network the object reference originated;

a name tag associated with the identity of a respective interceptor means; and

check data which is representative of the object reference, the identifier and the name tag.

23. A gateway according to any one of claims 1 to 12, further comprising interceptor locating means which is configured for determining, on the basis of an object reference, the location of the respective interceptor means for the referenced object.

24. A gateway according to claim 23, wherein the interceptor locating means is configured for invoking a respective interceptor means in the event one is not already running.

25. A gateway according to any one of claims 12 to 22, further comprising interceptor activating means which is configured for determining, on the basis of an object reference, whether there exists a respective interceptor means for the object reference.

26. A gateway according to claim 25, wherein the activating means is configured for dynamically generating a respective interceptor means when such an interceptor means does not exist.

27. A gateway according to claim 26, wherein the activating means is configured for retrieving for the referenced object an interface definition to be used for generating the interceptor means.

28. A gateway according to claim 27, wherein the activating means is configured for retrieving from the referenced object the interface definition.

29. A gateway according to claim 27 or claim 28, wherein the activating means is further configured to pass the retrieved interface definition to a compiler means so that the compiler means can generate an interceptor means for the respective referenced object.

30. A gateway for being situated between a first network and a second network, comprising:

first interface means for receiving a message (q), including a request for some information relating to an object;

interceptor means for passing the request to a repository containing respective information about objects and receiving a reply from the repository; and

second interface means for returning a message (q') to the originator, including information contained in the reply.

31. A gateway according to claim 30, wherein the first interface means is configured for receiving a request including an object name, and the second interface means is

configured for returning a message (q') to the originator including an object reference for the object name.

32. A gateway according to claim 31, wherein the interceptor means is configured for passing a request to a Naming Service.

33. A gateway according to claim 30, wherein the first interface means is configured for receiving a request including an object reference, and returning a message (q') to the originator including an object location for the object reference.

34. A gateway according to claim 33, wherein the interceptor means is configured for passing a request to a object locating service.

35. A gateway according to any one of claims 30 to 34, wherein, in the event the repostory does not return the requested information, the second interface means is configured for returning a message (q') to the originator including an exception.

36. A gateway according to any one of the preceding claims configured for operation within a trusted operating system environment, which supports Mandatory Access Control.

37. A gateway according to claim 36, including a trusted relay process which has the privileges necessary to pass messages between an inside compartment and an outside compartment of the trusted operating system, and wherein messages associated with a network inside of the gateway and the interceptor means are associated with the inside compartment, and messages associated with a network outside of the gateway are associated with the outside compartment.

38. A gateway according to claim 37, wherein the trusted relay process comprises the first interface means or the second interface means depending on which of the first or second networks is the inside network.

39. A gateway according to claim 38 or 37, as dependent on claim 6 or claim 19, wherein the secret is stored by the gateway in a further compartment, and wherein the trusted relay process has the privileges necessary to retrieve from the further compartment the secret in order to enact a hash operation.

40. A gateway according to claim 39, wherein the trusted relay process is configured to access the secret compartment to retrieve the secret in order to pass the secret to the interceptor means to enact the hash operation.

41. A gateway according to claim 39, wherein the trusted relay process is configured to access the secret compartment to retrieve the secret in order to enact the hash operation.

42. A gateway comprising one or more features substantially as hereinbefore described or substantially as illustrated in the accompanying drawings.

43. A method of passing an object reference across a gateway, including the steps of:

detecting the object reference; and

replacing the object reference with a modified object reference including the original object reference.

44. A method according to claim 43, wherein the modified object reference refers to the gateway and includes the original object reference.

45. A method according to claim 44, wherein the modified object reference has the form of an IOR[host: port: key], where the key includes the original object reference.

46. A method of passing an object invocation across a gateway, the method including the steps of:

receiving the object invocation;

extracting from the invocation an object reference; and

passing data associated with the invocation to the object referenced by the object reference.

47. A method according to claim 40, further including the step of verifying that the object reference is valid and that the object is available.

48. A method of publishing the availability of an object across a gateway, including the steps of:

generating an interceptor to receive messages including a request for some information relating to an object;

passing the request to a repository, which contains information relating to objects which are available, for the requested information and receiving a response;

returning to the requestor a message including the response.

49. A method according to claim 48, further including managing the repository to include only information relating to object that are available across the gateway.

Passing object references out from the gateway

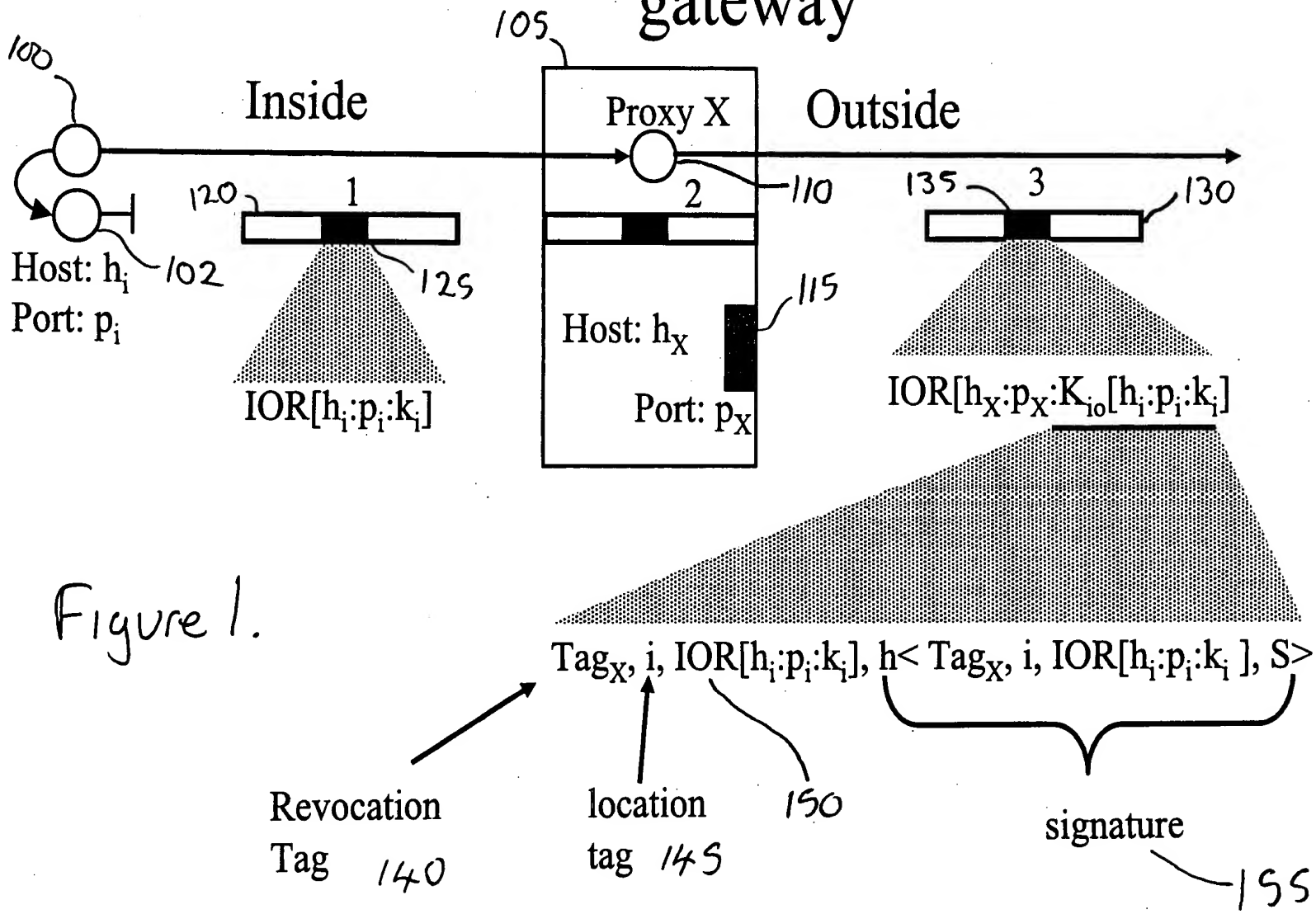


Figure 1.

Passing object references into the gateway

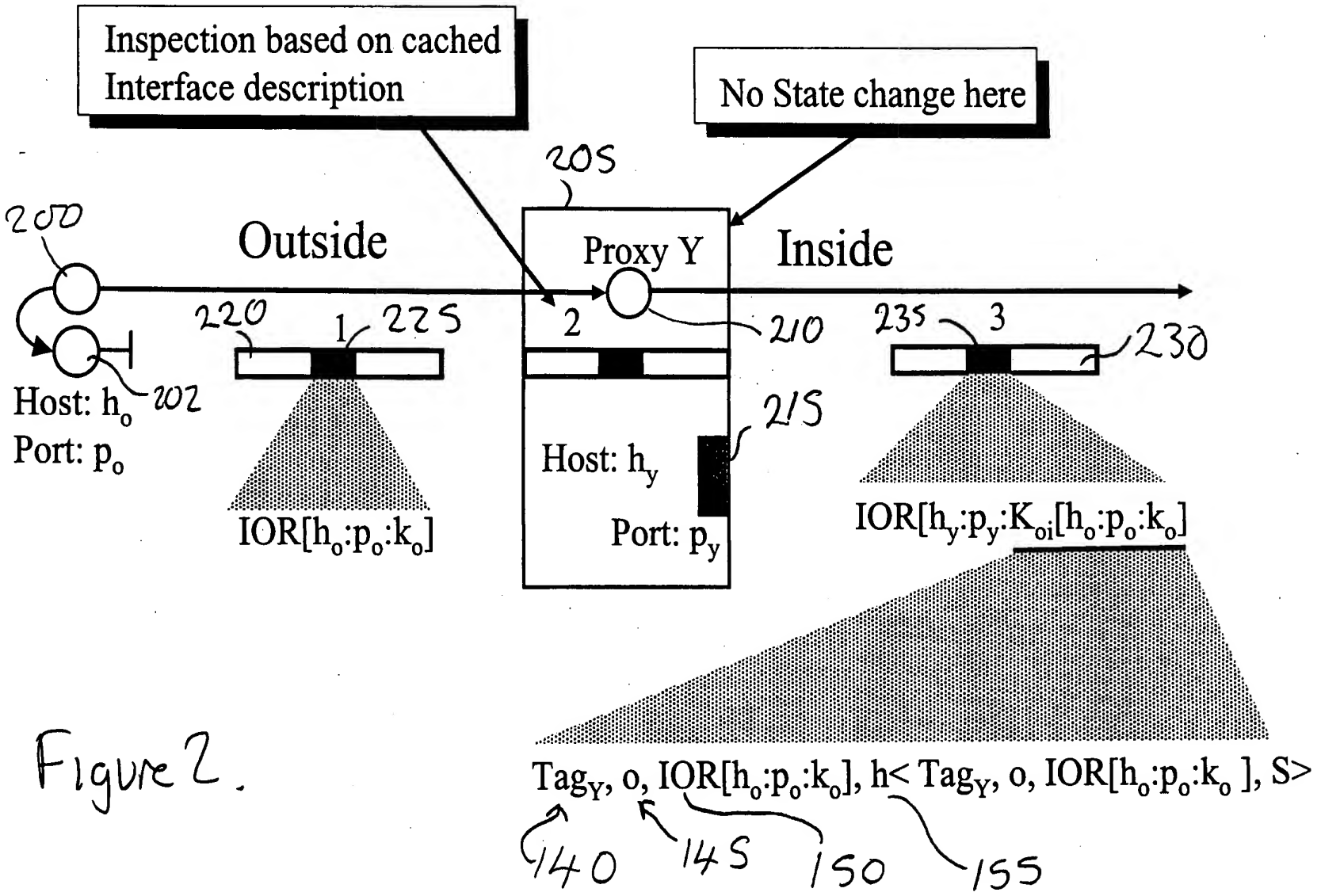


Figure 2.

Constructing a proxy dynamically

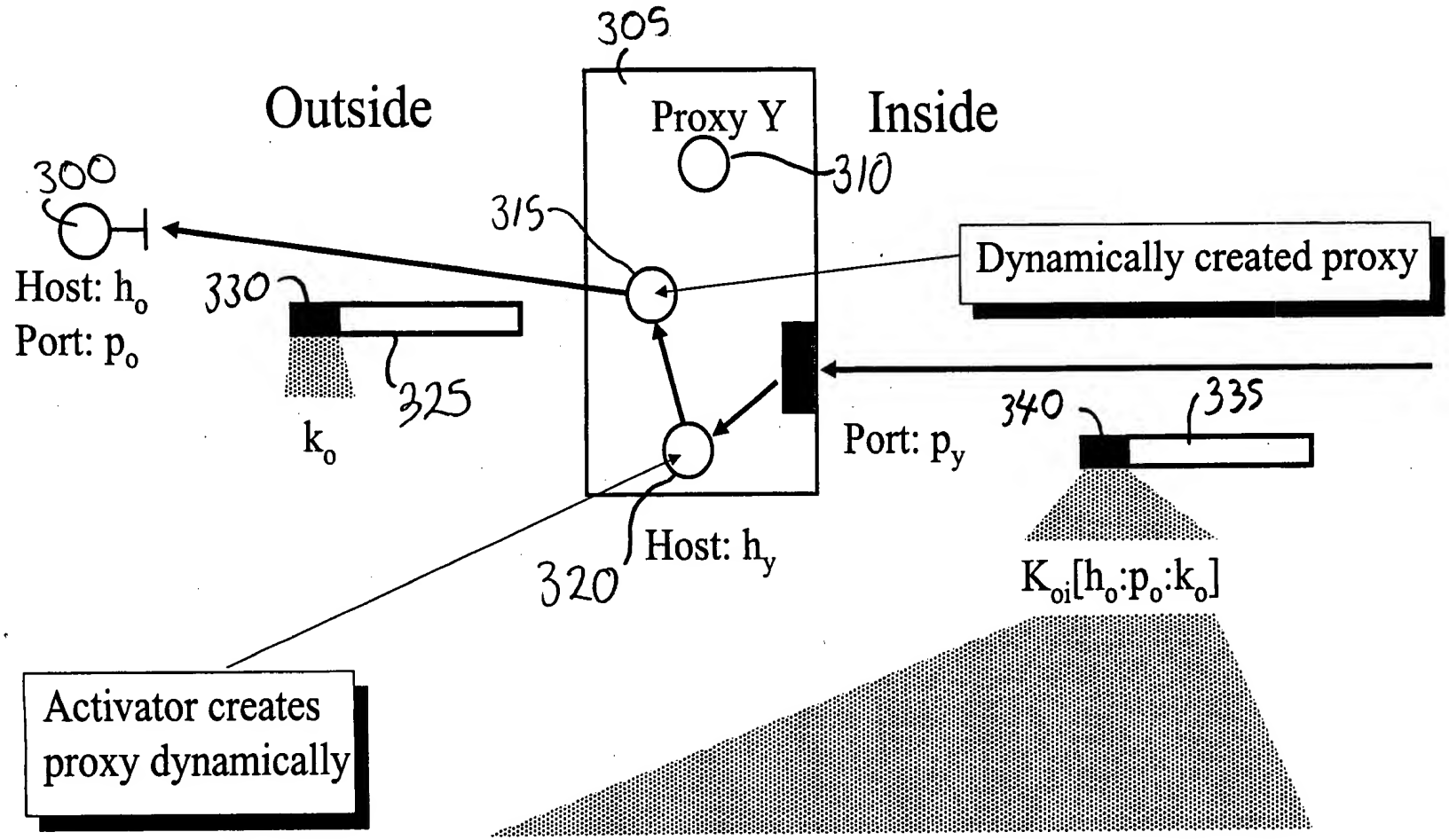
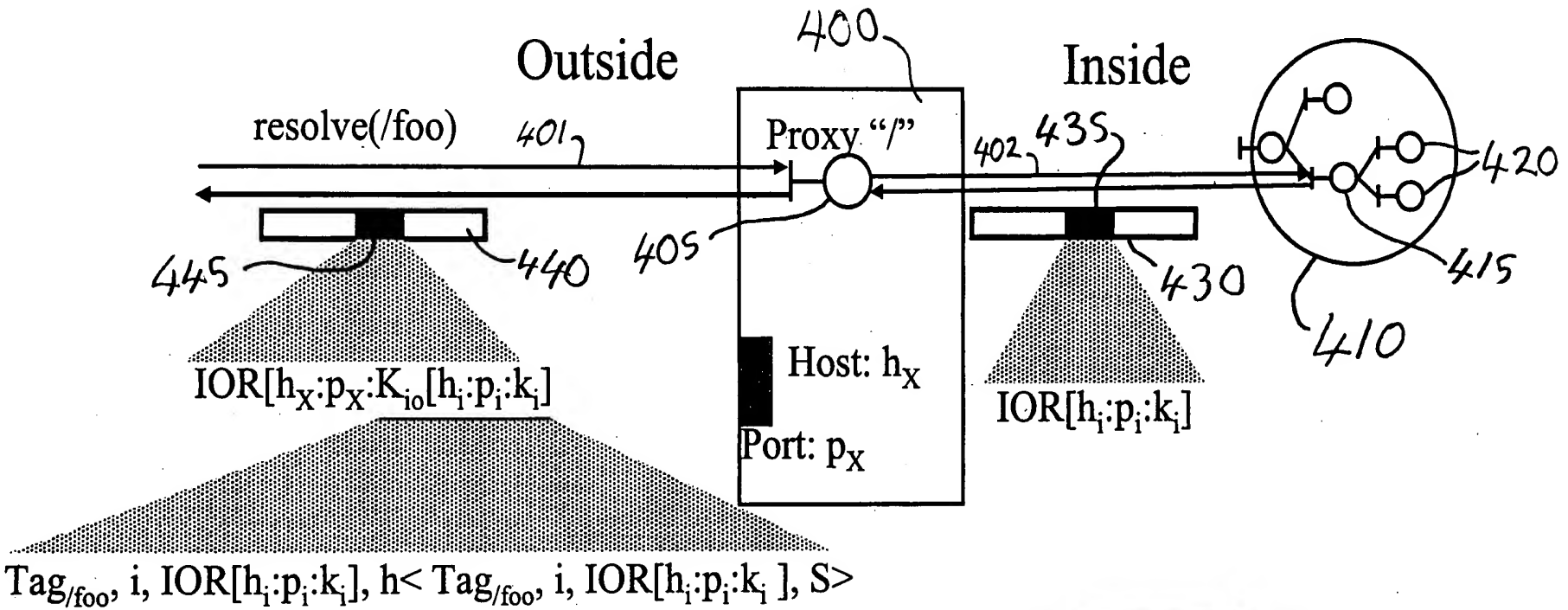


Figure 3.

Tag revocation (server side)



- Naming proxies generate new tags for "resolve"
- Tags are the full compound name
- Removing the entry in the Name Server invalidates the tag
- Ordinary proxies don't generate new tags

Figure 4

Use of location tags

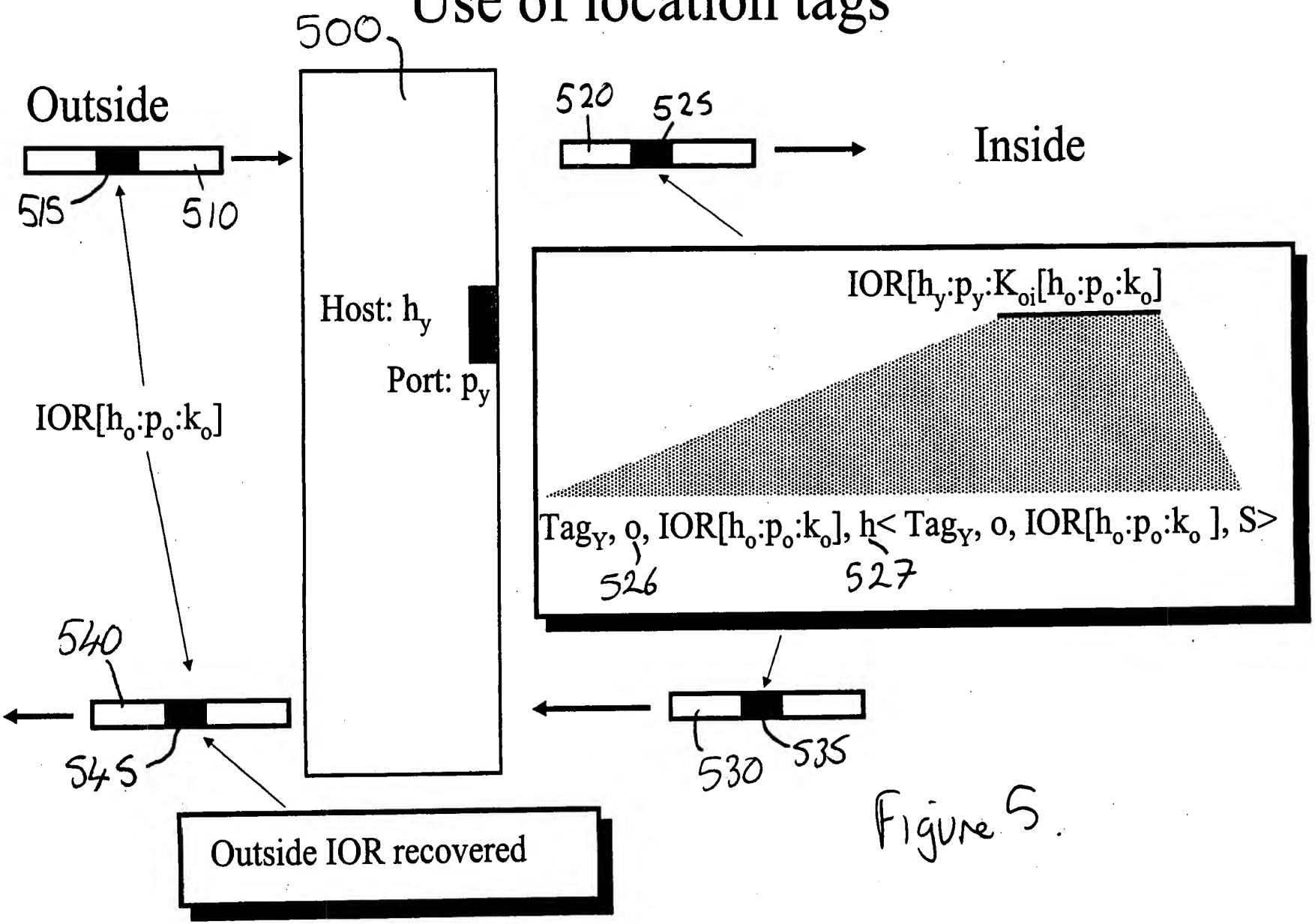


Figure 5.

Use of location tags

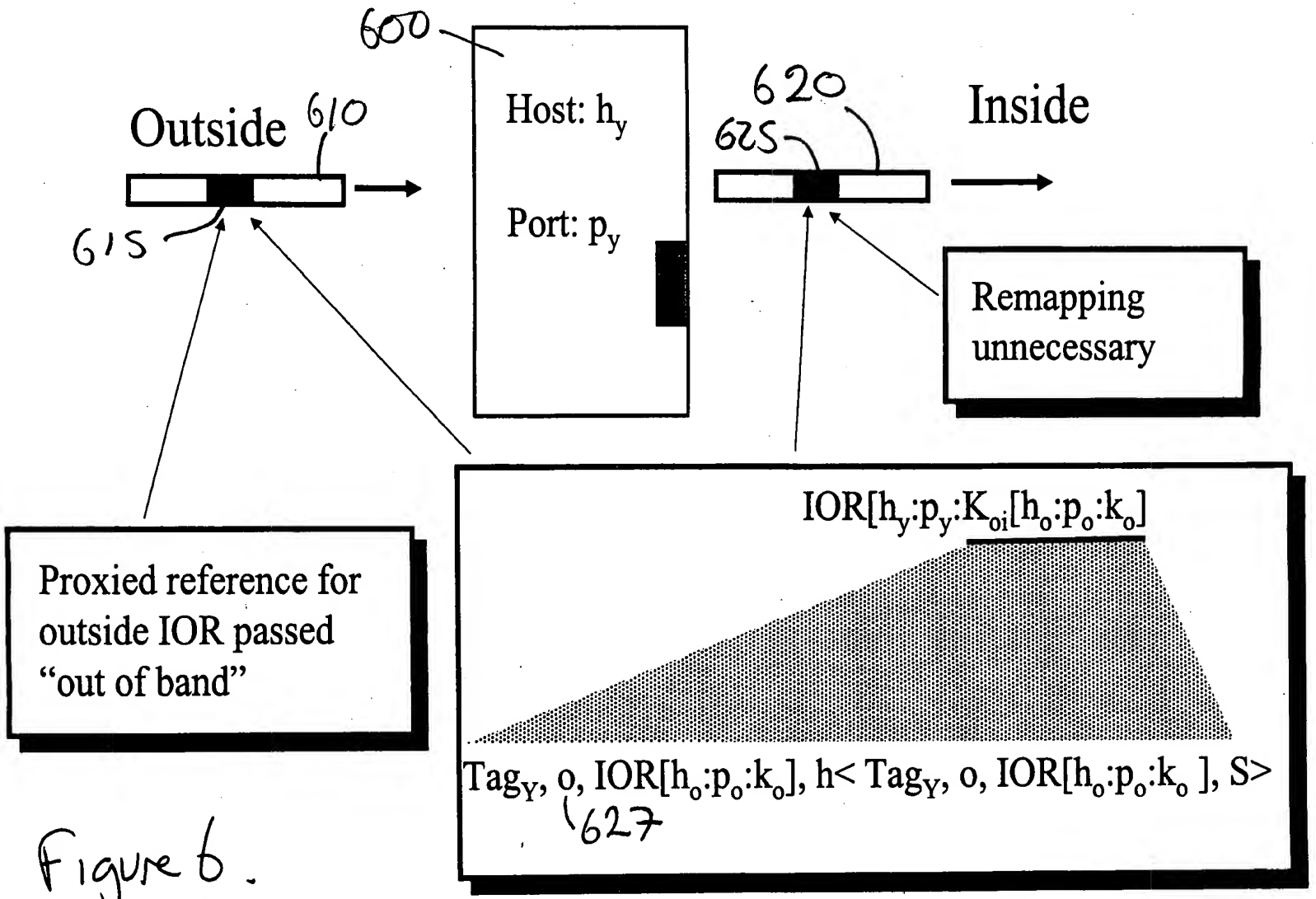


Figure 6.